Escuela Politécnica Superior

Reconstrucción programática de topologías de redes de transmisión a partir de la configuración de sus elementos

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Autor:
Juan Carlos Hernández Hernández

Tutor/es:
Ginés Fernández Costa

Julio 2021

Universitat d'Alacant
Universidad de Alicante

Resumen

El funcionamiento eficiente de las redes de transporte de un operador de telecomunicaciones requiere de un conocimiento previo de su topología física y lógica. Conocer y planificar estas primero, ayuda después a las tareas de configuración de los equipos, la ingeniería de tráfico, la segmentación de la red, entre otras labores típicas de estos escenarios. En redes extensas, las conexiones entre *routers* sufren con relativa frecuencia incidencias que afectan a los servicios finales, y que se traducen en cambios de estado de la red como conjunto.

Los ingenieros para enfrentar estos problemas cuentan con herramientas de monitorización. Estas permiten a los equipos de operación comparar el estado actual con la red inventariada y planificada, detectando de esta manera inconsistencias y agilizando la toma de decisiones para la restauración de los servicios que hacen uso de la misma. Es tendencia que cada fabricante de equipos tenga su herramienta específica para sus dispositivos. Esto hace la misión más difícil ya que en la actualidad por razones obvias, la flota de dispositivos de un operador es de composición variada, multifabricante. Por tanto sería necesario contar con un software de monitorización por cada fabricante de equipos que se tengan en la red.

Por esta razón el trabajo realizado se ha enfocado en el desarrollo de una aplicación que permita reconstruir la topología física de redes de transmisión a partir de la configuración de los elementos de la misma. Además la herramienta contempla una solución a la monitorización de redes multifabricante con la utilización de la biblioteca Napalm como capa de abstracción a la tecnología subyacente en los equipos, una especie de paraguas que permite hablar en un idioma universal con dispositivos de distinta procedencia.

Abstract

The efficient operation of the transport networks of a telecommunications operator requires prior knowledge of their physical and logical topology. Knowing and planning these first, helps with the tasks of equipment configuration, traffic engineering, network segmentation, among other typical tasks in these scenarios. In large networks, the connections between routers suffer relatively frequent incidents that affect the end services and cause changes in the state of the network as a whole.

To deal with these problems, engineers rely on monitoring tools. These allow operations teams to compare the current state with the inventoried and planned network, thus detecting inconsistencies and speeding up decision-making for the restoration of the services that make use of it. It is a tendency for each equipment provider to have its own specific tool for its devices. This makes the mission more difficult, because monitoring software would be needed for each equipment manufacturer in the network.

For this reason, the work carried out has focused on the development of an application that allows the physical topology of transmission networks to be reconstructed from the configuration of its elements. The tool also provides a solution to the monitoring of multi-manufacturer networks with the use of the Napalm library as an abstraction layer to the subjacent technology in the equipment.

Agradecimientos

En primer lugar agradecer infinitamente a mi familia por el apoyo para llegar hasta aquí. Sin lugar a dudas son la razón de ser del sacrificio realizado, dos años ya sin sentir su calor es mucho tiempo. A mis padres por su educación, su cariño, su comprensión. A mi madre donde quiera que este, dedicarle de manera especial este logro. A Soni mi pareja, han sido tiempos difíciles y sin ti no habría sido posible. A los amigos de siempre, Lucio, Víctor, ... y a los que esta excelente experiencia me ha dejado.

No podría faltar el agradecimiento a mi tutor Ginés por confiar en mí para este trabajo. Aunque los tiempos que corren imposibilitó un poco los encuentros presenciales, siempre sentí su apoyo y guía.

No podía faltar, a la Universidad de Alicante por su iniciativa de lanzar becas para cursar másteres oficiales a jóvenes latinoamericanos como yo.

"Nunca consideres al estudio como una obligación,
sino como una oportunidad para penetrar
en el bello y maravilloso mundo
del saber".

Albert Einstein.

Índice general

Índice de figuras

1 Capítulo 1: Introducción

1.1. Motivación

Manejar las redes y servicios de telecomunicaciones de forma económica, fiable y eficiente ha sido el principal objetivo de toda empresa operadora desde los inicios de los sistemas de telecomunicaciones. Tan pronto se dispuso de ordenadores, se desarrollaron aplicaciones destinadas a aumentar los niveles de eficacia para además reducir el coste de las operaciones mediante la automatización de una serie de procedimientos comerciales y administrativos, así como de operaciones, por ejemplo la facturación, la gestión de órdenes de servicio, de fallos, alarmas, etc [1], [2], [3].

Hoy en día, un operador de telecomunicaciones se enfrenta a muchos retos, dada la necesidad de suministrar servicios diversificados, digitales y convergentes en un entorno de red multitecnológico y multifabricante, que cambia rápidamente. Es esencial que estas entidades encargadas respondan rápidamente a los cambios del mercado y que la tecnología, satisfaga las necesidades de los clientes y reduzca los gastos operativos (OPEX) [4],[5].

Para abordar estas cuestiones, los sistemas OSS (*Operations Support Systems*) del operador de telecomunicaciones deben cumplir los siguientes requisitos: tener capacidad de aprovisionamiento, que sea automatizado para una rápida prestación de servicios, posibilite una supervisión proactiva y reactiva que garantice la calidad de extremo a extremo, gestionar eficiente y de manera eficaz los problemas, y por último que presente gran flexibilidad de

personalización y ajuste para ofrecer nuevos productos al mercado a tiempo [4].

Por esta razón el trabajo de investigación realizado se enfoca en el desarrollo de un herramienta que sea capaz de hacer frente a los retos en los que se ve envuelto un operador de telecomunicaciones a la hora de gestionar y monitorizar sus redes. Asumiendo un escenario de equipamiento de redes de diferentes fabricantes con una topología física variada, que a su vez sea escalable y permita hacer frente a las tareas de operación y mantenimiento de manera eficiente, que se traduzca en una disminución de las interrupciones en el servicio al cliente final.

Agregar que esta investigación da un enfoque integrador a las tecnologías emergentes y afianzadas en el mundo TI (Tecnologías de la Información) aplicadas al desarrollo de software para la industria de las telecomunicaciones como pueden ser: la automatización de tareas de red, el desarrollo en python, el despliegue en la nube, el trabajo con servicios web y visualización de la información en forma de aplicación accesible desde internet.

1.2. Objetivos

La operación eficiente de redes de transporte requiere de un conocimiento previo de su topología. En redes extensas, las conexiones entre *routers* sufren con relativa frecuencia incidencias que afectan a los servicios finales, y que se traducen en cambios de estado de la red como conjunto.

La recreación programática de la topología física de la red permite a los equipos de operación comparar el estado actual con la red inventariada y planificada, detectando de esta manera inconsistencias y agilizando la toma de decisiones para la restauración de los servicios que hacen uso de la red.

En este trabajo se realiza un estudio de las tecnologías utilizadas para dar solución a los retos

que presentan los operadores de telecomunicaciones para la gestión y mantenimiento de sus redes. Se desarrollará una herramienta web que es capaz de reconstruir la topología física de la red de transporte desplegada de dicho operador a partir de la configuración de sus elementos, para la cual es indiferente la tecnología o proveedor de equipamiento (más conocido como marca o fabricante) subyacente en el escenario. La base de nuestro trabajo lo constituye una librería de código abierto (NAPALM) que sirve como una capa de abstracción a la tecnología base de los *routers* del operador.

Como colofón demostraremos su funcionalidad para un entorno de topología "real" en el software de simulación de red del fabricante.

Es por eso que los objetivos principales que nos hemos propuesto con este trabajo han sido:

- Realizar una revisión del estado del arte con el propósito de conocer la situación actual en este tipo de aplicaciones.
- Generar una topología realista dentro del emulador eNSP de Huawei,
- Desarrollar un *script* de captura de datos con NAPALM para la red en tiempo real.
- Procesar y mostrar la información de la topología en forma de una aplicación web desplegada en la nube.

2 Capítulo 2: Marco Teórico

2.1. Evolución de la arquitectura de gestión de redes

No hace mucho tiempo desde que la operación y mantenimiento (O&M) de las grandes redes, como pueden ser las de un operador de telecomunicaciones, se hacía de forma monolítica. Los problemas de red se detectaban cuando ya era demasiado tarde. El operador del NOC (*Network Operations Center*) intuía qué sistemas estaban relacionados con el problema y enviaba a sus técnicos o personal encargado, uno por uno analizando los *logs* de cada uno de los elementos hasta encontrar el origen del problema. Esto dificultaba enormemente el trabajo del operador e incrementaba drásticamente el MTTR (*Mean Time To Repair* o Tiempo Medio de Reparación) [6].

En este momento es cuando surgen los EMS (*Element Management System*) y NMS (*Network Management System*) [1].Los primeros, son específicos de cada fabricante, gestionan los equipos de una única tecnología. Recopilan alarmas, inventario, datos de rendimiento e información de configuración. Los NMS, llamados comúnmente herramientas de monitorización, proporcionan una visión segmentada, y dado que estos no realizan todas las funciones del marco de referencia FCAPS (*Fault, Configuration, Account, Performance, Security*), nos encontramos con una gran variedad de herramientas, cada una de ellas enfocada en una función concreta [7].

Con la proliferación de nuevas redes de comunicaciones heterogéneas por ejemplo Móvil,

SDH, PON, FTTH, IP/MPLS, etc. el número de EMS creció [8]. Ante la gran variedad de NMS y EMS existentes en el mercado, cada uno dedicado o especializado en una determinada tecnología o capa de red, aparecieron herramientas "paraguas", encargadas de integrarse con el resto de NMS, EMS y centralizar la información en una única consola para facilitar el trabajo del operador y aportar una visión de servicio. Esto permitió la correlación entre las distintas capas de red y la capacidad de relacionar un problema final con el origen (RCA o *Root Cause Analysis*), aun cuando son elementos de diferentes tecnologías. A la vez, los operadores pudieron reducir el MTTR ante fallos y caídas de red, mejorando la experiencia de usuario e indirectamente incrementando los beneficios [6].

Podemos ver los sistemas de soporte a las operaciones (OSS) como la capa que aporta visión de servicio al conjunto de elementos EMS y NMS. En el nivel más alto de estos, la capa de soporte a los sistemas de negocio (BSS), es desde la que se gestionan las diferentes órdenes. El aprovisionamiento se realiza a través de los sistemas BSS y atraviesa cada una de las capas hasta llegar a los elementos de red (NE o *Network Elements*). De manera inversa, se asegura la calidad del servicio, empezando por la monitorización de los recursos de red y subiendo cada una de las capas hasta obtener una visión de servicio [9].

En la figura 2.1 se hace referencia, entre paréntesis, a las diferentes capas definidas por la ITU (*International Telecommunication Union*) en el modelo TMN (*Telecom Management Network*): BML (*Business Management Layer*), SML (*Service Management Layer*), NML (*Network Management Layer*), EML (*Element Management Layer*) y NEL (*Network Element Layer*) [11], [10].

En esta evolución, el próximo paso se da en la automatización con entornos de contenedores, redes SDN (*Software Defined Networks*) y la virtualización de las funciones de red (NFV, *Network Function Virtualization*), cobrando gran importancia, tanto en la arquitectura, como en la automatización del despliegue. El término que agrupa todo esto más conocido como orquestación, se hace más popular hoy en día.

La arquitectura de gestión, funcionalmente, cumple su razón de ser en el NOC, sitio desde

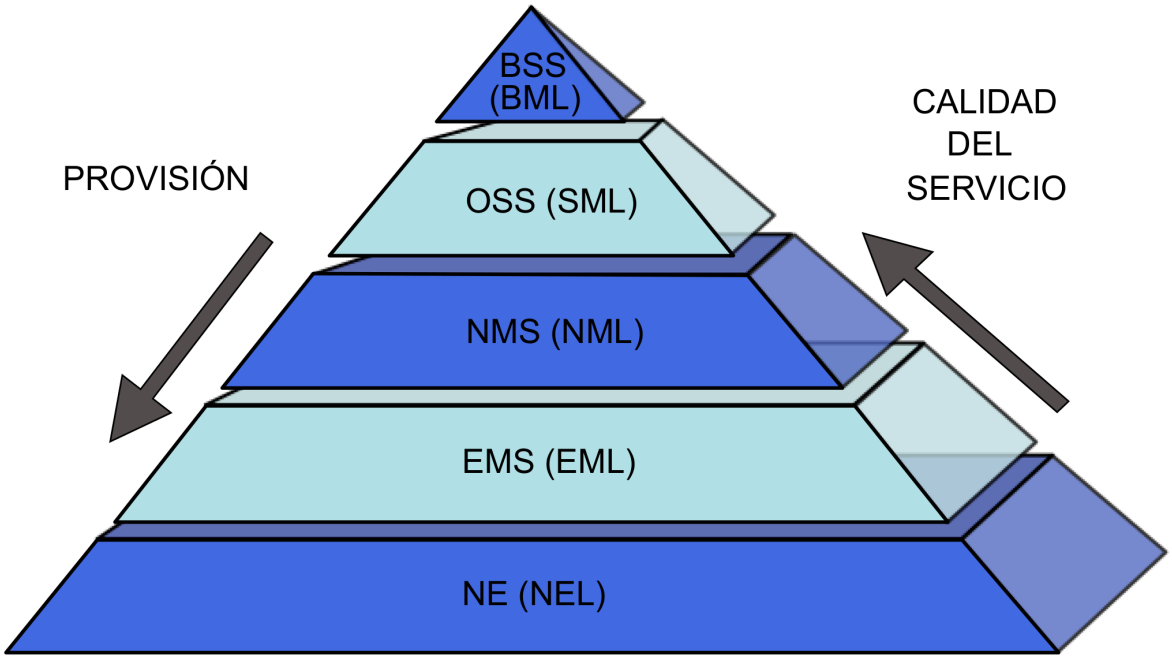


Figura 2.1: *Diagrama de relación BSS/OSS.*

(Tomado de [10])

el cual se realizan las tareas de control de la red. En él, el personal especializado, se encarga de diseñar, instalar, dar mantenimiento correctivo y preventivo a la operación (gestión, soporte y monitorización) de redes de telecomunicaciones. Aquí llevan a cabo la monitorización en función de alarmas o de condiciones que necesiten atención especial para prevenir fallos en el rendimiento de las redes y con ello al servicio prestado a los clientes finales.

2.2. Sistemas de monitorización de red

El término "monitorización de la red" describe un sistema que supervisa continuamente toda la topología de la red para detectar interferencias, ralentizaciones o fallos en los componentes y notificar al responsable de la red por correo electrónico, SMS, por una interfaz gráfica u otras alarmas en caso de cualquier problema [12]. La monitorización suele estar asociada a las funciones de gestión de la red [13]. La gestión es necesaria para garantizar que la red esté en funcionamiento [14]. El sistema ideal para estos fines debería tener las siguientes propiedades:

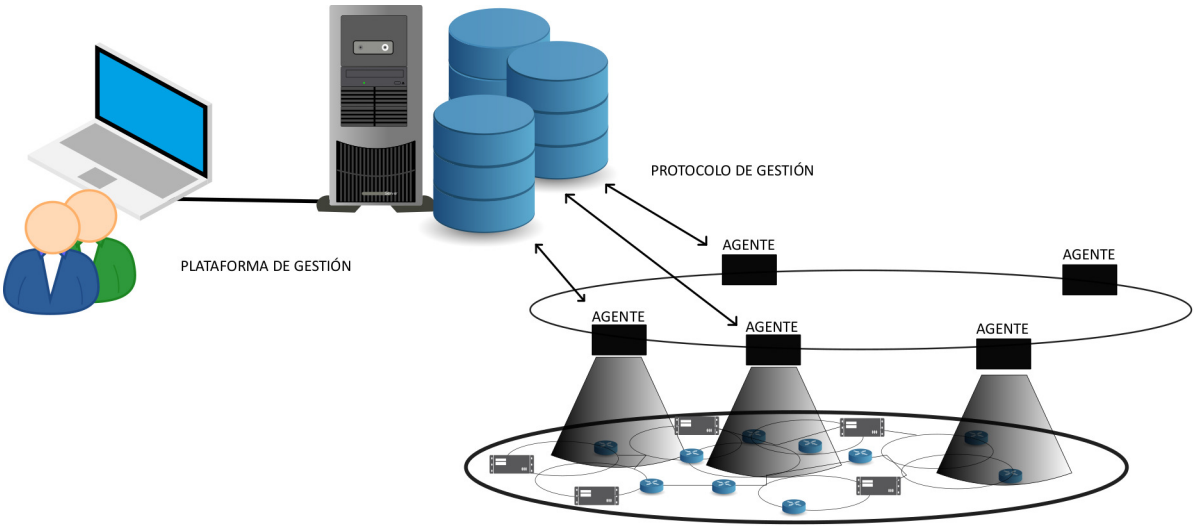


Figura 2.2: *Arquitectura de un sistema de gestión de red.*

- Debe ser automático y supervisar continuamente la red.
- Debe informar rápidamente al personal encargado sobre el problema tan pronto como surja.
- Debe ser lo suficientemente inteligente como para señalar el problema y su ubicación exacta en la topología de la red.
- Debe proporcionar autenticación remota y autorización para que el administrador pueda acceder al sistema de monitorización desde cualquier lugar.

La importancia y relevancia de las características de la red varía según las aplicaciones, así como los parámetros fundamentales a supervisar. Una red de distribución de contenidos puede estar interesada en medir latencia desde sus clientes hasta el servidor de aplicaciones, mientras que una aplicación de equilibrio de carga puede tener el ancho de banda disponible como un criterio útil para ajustar la carga en los servidores. Aunque las más estudiadas son la latencia, la pérdida de paquetes, el ancho de banda y la detección de caminos; otras características de la red, como la congestión, el retardo de las colas, el descubrimiento de la topología y la determinación del cuello de botella también tienen gran importancia.

En nuestro trabajo utilizaremos el criterio del descubrimiento de la topología de la red como base para el análisis de fallos de más alta en complejidad que se puedan dar.

2.2.1. Tipos de monitorización de la red

Existen dos enfoques básicos para la monitorización de la red, la monitorización activa y la monitorización pasiva [15].

Monitorización activa: consiste en inyectar sondas en la red, específicamente con el propósito de la monitorización. La monitorización activa tiene el coste de enviar tráfico adicional en la red; sin embargo, en la mayoría de los escenarios, el tamaño de los paquetes es relativamente pequeño comparado con la capacidad real de la red, por lo que el coste de inyectar tráfico adicional es mínimo. Este coste puede reducirse disminuyendo la tasa de sondeo; aunque esto puede reducir la calidad de las características medidas.

Monitorización pasiva: es el proceso de observar el tráfico de red existente y recoger información de él sin inyectar sondas de monitorización adicionales en la red. En un sistema basado en mediciones pasivas, la información de medición de la red se recupera a través de los paquetes que se envían como parte de otra aplicación. Los paquetes son capturados por la aplicación de monitorización, que se despliega en los *hosts* de origen y destino.

El modo pasivo de monitorización evita la sobrecarga de introducir tráfico de medición en la red y también evita el uso de datos obsoletos para las mediciones. Sin embargo, en el modo pasivo, la aplicación de monitorización tiene poco o ningún control sobre el intervalo de sondeo, los paquetes y la velocidad de la red, el tamaño de los paquetes o la ruta que se va a supervisar. Además, debido al aumento de la capacidad de los enlaces centrales, es costoso identificar y medir el paquete de un flujo concreto. Por último y no menos importante, la monitorización pasiva podría plantear problemas de privacidad.

La preferencia por la forma de realizarlo varía según la aplicación. Por ejemplo, si queremos

medir el rendimiento de un sistema de detección de intrusos podría preferir un enfoque pasivo, mientras que una aplicación que emplee medidas de mejora del rendimiento y supervisión de la topología de una red, puede desear un esquema activo. Se puede también emplear una combinación de los esquemas pasivos y activos para una mayor eficiencia.

2.2.2. Tecnologías de monitorización

A pesar de lo que los fabricantes de herramientas de monitorización hagan creer, solo un número finito y limitado de tecnologías puede ser utilizado para monitorizar. La sofisticación viene dada por la frecuencia, agregación, relevancia de las pantallas y la facilidad de la implementación [16]. Algunas de estas tecnologías se exponen a continuación.

SNMP (*Simple Network Management Protocol*): se compone de una lista de elementos que devuelven datos sobre un dispositivo en particular. Puede ser, por ejemplo, la CPU o la media de bits por segundo transmitidos en los últimos cinco minutos. SNMP proporciona datos basados en un disparador (cuando uno de los puntos de datos internos cruza un umbral) o una solicitud de sondeo SNMP [17].

SNMP permite el acceso a la MIB (*Management Information Base*) que contiene los objetos OID (*Object Identifier*) susceptibles de ser gestionados dentro de los equipos activos de la red [18] y al protocolo RMON (*Remote Monitoring*) versión 1 (esta es básicamente una extensión de la MIB del protocolo SNMP).

Telemetría: un agente de software instalado en un elemento de la red permite la transmisión automática de indicadores clave de rendimiento en tiempo real. La telemetría está sustituyendo rápidamente a SNMP, porque es más eficiente, puede producir muchos más puntos de datos y es más escalable. Y los estándares de telemetría, como NETCONF/YANG [19] [20], están ganando adeptos como forma de ofrecer el mismo soporte multiproveedor que SNMP [21], dada sus carencias para efectuar configuraciones especializadas y complejas, como por ejemplo el uso de UDP (*User Datagram Protocol*) como protocolo de transmisión de datos. Aunque

NETCONF/YANG requiere de una arquitectura "cliente-servidor" que para la actualidad es un poco rígida.

ICMP (*Internet Control Message Protocol*): es utilizado por dispositivos de red como *routers* y *switches* para enviar mensajes de error indicando que un *host* no es alcanzable junto con algunos otros diagnósticos [22].

Syslog (*System Logging Protocol*): se trata de un protocolo estándar utilizado para enviar mensajes de registro o eventos del sistema a un servidor específico, llamado servidor de syslog. El Syslog se utiliza principalmente para recopilar varios registros de los dispositivos de diversas máquinas diferentes en una ubicación central, para la supervisión y análisis. Un servicio Syslog o agente toma los eventos que ocurren en el dispositivo y los envía a un sistema remoto de escucha (servidor de destino syslog) [23].

El protocolo está habilitado en la mayoría de equipos de red, como *routers*, *switches*, cortafuegos e incluso en algunas impresoras y escáneres. Además, Syslog está disponible en sistemas basados en Linux/Unix y en muchos servidores web como Apache [24].

IP SLAs (*Internet Protocol Service Level Agreements*): son un conjunto bastante completo de capacidades integradas en los equipos de Cisco (y de otros fabricantes). Estas capacidades se centran en garantizar que el entorno de la WAN, y más concretamente de la VoIP, es saludable al utilizar los dispositivos que forman parte de la infraestructura de la red en lugar de requerir que se configuren dispositivos separados para ejecutar pruebas [16].

Scripts: la ejecución de un *script* para recoger información puede ser tan simple o complicado como el desarrollador decida realizarlo. Además, el *script* puede ser ejecutado localmente por un agente en el mismo dispositivo e informar del resultado a un sistema externo. O bien, puede ejecutarse remotamente con privilegios elevados [16].

Esta última variante es la utilizada en nuestro trabajo, por lo que la detallamos más adelante.

2.2.3. Herramientas contemporáneas de monitorización de redes

La información relativa al estado genérico de la red completa, para detectar las áreas que fallan o que necesitan ser gestionadas, son recogidas por las herramientas de monitorización de la red. Estas son capaces de comprobar el rendimiento total en comparación con una red modelo de referencia y/o inventariada en la que todo funciona perfectamente. Utilizando su ayuda, los administradores de red observan sin esfuerzo el rendimiento y el funcionamiento de sus infraestructuras. Estas herramientas también proporcionan información para otros análisis como puede ser en el área de la ingeniería de tráfico [25].

El mercado de este tipo de aplicaciones está dominado en primer lugar por los proveedores de equipamiento. Estos proveedores sacan como un producto más al mercado sus sistemas de gestión y monitorización para sus equipos. Esta situación no es compatible con la dinámica de crecimiento y desarrollo de un operador, ya que es usual por varias razones, no depender de un único suministrador de equipamiento.

Por otro lado se encuentran las empresas desarrolladoras de software para redes o tecnologías de la información (TI). Estas empresas utilizan el enfoque general de otros sistemas, los mejoran, aplican las tecnologías de gestión de redes necesarias y le ofrecen a sus clientes (los operadores) una aplicación a medida de sus necesidades. Es por eso que en el mercado nos podemos encontrar soluciones de código abierto o propietarias.

Ahora, sea cual sea el modelo y/o herramienta elegida, o la magnitud que requiera lo que se desea conseguir para la monitorización, deben ser capaces de realizar algunas o todas las tareas que se mencionan a continuación:

- Detección de fallos y alertas.
- Detección de dispositivos.
- Predicción de la tendencia del tráfico de la red.

- Configuración de la red.

- Generación de mapas topológicos de la red.

Se ha demostrado que no existe una herramienta de monitorización perfecta que satisfaga todas las necesidades de cada organización. Cada herramienta tiene sus propias características especiales que la hacen diferente de la otra y depende totalmente de los requisitos para seleccionar una herramienta de monitorización u otra [26].

El monitor de rendimiento de red Orion de SolarWind cubre todos los componentes principales necesarios para la monitorización, pero tiene una interfaz gráfica de usuario compleja que hay que aprender a utilizar correctamente y además es costosa [26].

Nagios también es una muy buena herramienta y es de código abierto como OpenNMS, por lo que podemos personalizarla de acuerdo a las necesidades específicas de la organización, así como crear nuestros propios *plugins* de las características que necesitemos añadir e integrarlos a Nagios, pero se necesitan expertos y tiempo de desarrollo para personalizar la herramienta según los requisitos, lo que puede aumentar el coste. Brinda la posibilidad también, de emitir alertas encapsuladas [26].

NetBrain es otra herramienta que es muy adecuada para la monitorización. Es buena en las organizaciones que estudian la red y realizan análisis sobre el tráfico de la red o la predicción de tendencias, ya que proporciona todas estas características. NetBrain se puede utilizar en organizaciones donde la mayoría de las tareas están automatizadas, ya que proporciona muchas funciones de automatización [26].

PRTG (*Paessler Router Traffic Grapher*) se puede utilizar cuando se requiere una supervisión distribuida o cuando se requiere la visualización de la red, ya que proporciona mapas geográficos en tiempo real [26].

Agregar que de estas herramientas existen muchas más, tantas como variedad hay en el mercado de empresas dedicadas al desarrollo de estos software, que por demás tienen un

precio alto de sus productos [27]. Muchas veces para aplicaciones puntuales, lo oportuno sería que los administradores creen sus propios programas.

2.2.4. Monitorización de redes en ambientes *multivendor*

En las redes de gran magnitud, es casi un estándar contar en ella elementos de varios fabricantes (ambientes *multivendor*) [25]. Cuando esto sucede la gestión se convierte en una labor compleja y que necesita varios factores para llevarla a cabo al no poder contar con una herramienta capaz de agrupar los equipos de todos los fabricantes.

Como paso esencial en nuestro trabajo, nos dimos a la tarea de indagar sobre la existencia de investigaciones o proyectos que guardaran relación con este tema especifico, también buscamos en base de datos indexadas todo ello sin obtener resultados positivos. Percatarnos que falta por hacer en este campo, sin dudas que constituyo un aspecto motivador. No obstante, resaltamos que en el mercado del desarrollo de software para la industria TI existen herramientas que permiten gestionar y monitorizar las redes de datos en ambientes *multivendor*, tales como Tivoli [28] de IBM; ProactiveNet Performance Manager [29] de BMC; Unified Infraestructure Management [30] de Broadcom; OP Manager [31] de ManageEngine; y Magellan NMS Network Management System [32] de Imagine Communications. Pero de manera general estas soluciones no son para nada baratas y se debe obtener una licencia para usarse en ambientes empresariales e incluso académicos. Considerar, además, que algunas herramientas de gestión y monitorización de algunos fabricantes como Cisco, Hewlett Packard o Dell permiten administrar dispositivos de otros fabricantes, siempre y cuando se cuente con los OID en la MIB para cada caso en particular, lo que lo convierte en un proceso molesto y que emplea mucho tiempo para poder implementarlo.

Para el caso de las redes con tecnología de Huawei, el asunto es más complejo. La búsqueda en base de datos científicas no arrojó resultados, lo que quiere decir que sobre esto poco se ha hecho. Se deja la posibilidad de que este tipo de información este de alguna forma controlada

por las empresas dedicas al desarrollo de software de red (o para TI) por los dividendos que le pueden sacar dado el "secretismo" que ronda esta tecnología. Solo en un foro virtual de la multinacional hay una pista de que es posible. Un usuario destacado expone que si se quiere automatizar y/o extraer reportes del U2000/eSight (OSS para los equipos de Huawei) se puede invocar un API (*Aplication Programmable Interface*) vía un WebService que puede ser XML, YANG, REST; mediante el uso de TL1 (Transmission Language 1), al cual se accede por consola del servidor, y se puede comunicar con OLTs (*Optical Line Termination*), ONTs (*Optical Network Terminal*) y cualquier elemento de red del que se quiera obtener información [33].

Si a este tipo de desarrollo le aplicamos un procedimiento parecido para otra tecnología que tengamos en la red y con el apoyo de una buena herramienta como CORBA de habitual uso en estos escenarios, "permitiría que diversos componentes de software escritos en múltiples lenguajes de programación y que corren en diferentes computadoras, puedan trabajar juntos; es decir, facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos" [34].

2.3. Descubrimiento de la topología en redes IP heterogéneas

La topología de una red es clave para determinar su rendimiento. Se refiere a la forma en que los distintos nodos, dispositivos y conexiones de la red están dispuestos física o lógicamente en relación con los demás [35].

Existen numerosas formas de organizar una red, todas ellas con distintos pros y contras, y algunas son más útiles en determinadas circunstancias que otras. Los administradores tienen una gama de opciones cuando se trata de elegir una topología, y esta decisión debe tener en cuenta el tamaño y la escala de su negocio, sus objetivos y su presupuesto. Su gestión eficaz incluye varias tareas, como la gestión de la configuración, el mapeo visual y la supervisión

general del rendimiento [36].

Hay dos enfoques de la topología de red: el físico y el lógico.

- La topología física de la red, como su nombre indica, se refiere a las conexiones físicas y a las interconexiones entre los nodos y la red: cables, alambres, etc.

- La topología lógica de la red es un poco más abstracta y estratégica, y se refiere a la comprensión conceptual de cómo y por qué la red está dispuesta de la manera en que lo está, y cómo los datos se mueven a través de ella.

Existen varios tipos de topología de red y todos son adecuados para diferentes propósitos, dependiendo del tamaño total de la red y de sus objetivos como muestra la figura [37].

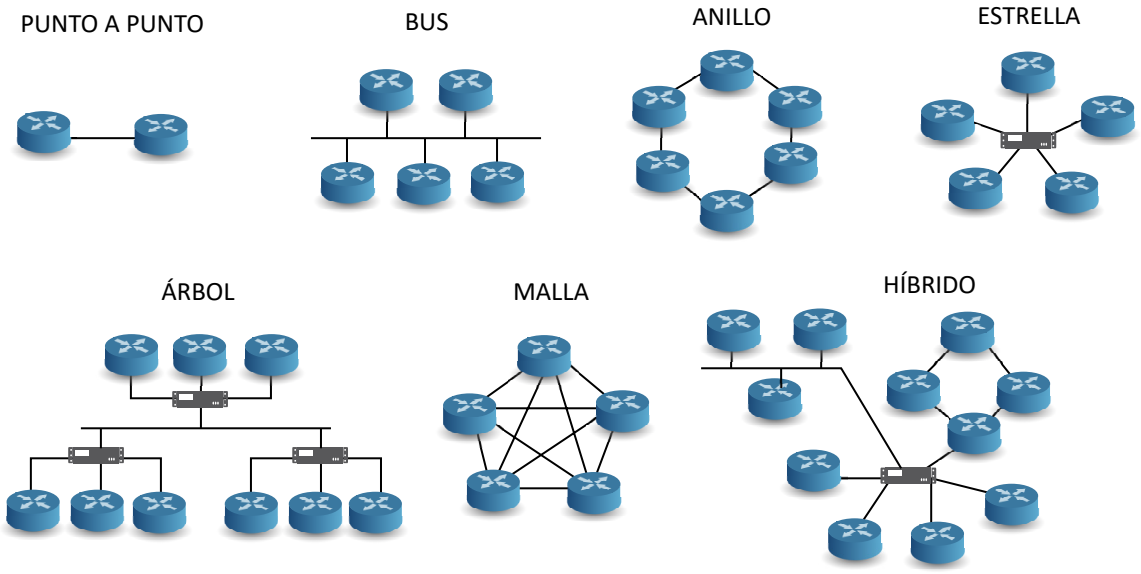


Figura 2.3: *Tipos de topologías físicas de red.*

(Elaborado a partir de [38])

El conocimiento de la topología física actualizada de una red IP es crucial para una serie de tareas críticas de gestión de la red, incluyendo la gestión reactiva y proactiva de los recursos, la correlación de eventos y el análisis de la causa raíz.

Dado la naturaleza dinámica de las redes IP de hoy en día, hacer un seguimiento de la

topología manualmente es una tarea desalentadora (si no imposible). Por lo tanto, se necesitan algoritmos eficaces para descubrir automáticamente la topología física de la red. Los trabajos anteriores se han centrado normalmente en [36]:

1. Descubrir la topología de capa 3 o capa de red, lo que implica que se ignora la conectividad de todos los elementos de capa 2 o de enlace según el modelo OSI (por ejemplo, conmutadores y puentes).

2. Soluciones propietarias dirigidas a familias de productos y requerimientos específicos.

Normalmente, un solo fallo en la red puede provocar una avalancha de señales de alarma procedentes de diferentes elementos de red interrelacionados. El conocimiento de las interconexiones de los elementos es esencial para filtrar las señales de alarma secundarias y correlacionar las alarmas primarias para localizar el origen del fallo en la red [36].

Además, un mapa físico completo de la red permite un análisis proactivo del impacto de los fallos de enlaces y dispositivos. La identificación temprana de los puntos críticos de fallo permite al gestor de la red mejorar la capacidad de supervivencia de la red (por ejemplo, añadiendo rutas de enrutamiento) antes de que se produzcan las interrupciones.

En consecuencia, la mayor parte del trabajo de mapeo de Internet se ha se ha concentrado en el descubrimiento automatizado de topologías de WAN y, más concretamente, en la topología de las interconexiones (es decir, la topología de capa 3) [36].

Descubrir una topología de capa 3 es relativamente fácil siempre que la información de enrutamiento estándar esté disponible, porque los *routers* deben conocer explícitamente a sus vecinos de capa 3 para poder realizar su función básica.

Algunos de los trabajos proponen una heurística para inferir la topología de la capa 3 empleando comandos ICMP básicos, como *ping* y *traceroute* [39]. La retroalimentación a salto proporcionada por el comando ICMP *traceroute* se emplea en el contexto del proyecto Mercator

[40], cuyo objetivo es descubrir la adyacencia entre *routers*. Más específicamente, Mercator descubre conexiones de *routers* adyacentes enviando comandos *traceroute* y *ping* desde la red y utilizando sólo sondas de saltos limitados; también emplea varias heurísticas inteligentes para descubrir el mapa de la capa 3. El proyecto Rocketfuel [41] utiliza los resultados de 294 servidores públicos de *traceroute* para construir topologías de ISP a nivel de *router*; también emplea técnicas que pueden reducir la cantidad de sondeo y resolver el *traceroute* de direcciones. Un problema relacionado es el de inferir la topología en capa 3 de los árboles de multidifusión IP. Las soluciones propuestas para este problema se basan en correlacionar las mediciones de multidifusión de extremo a extremo (por ejemplo, las características de pérdida de paquetes), o utilizando algún mecanismo especializado que requiera la cooperación de los *routers* [42]. Tutschku y Baier [43] utilizan medidas de tiempo de ida y vuelta obtenidas a través de paquetes ICMP para definir una métrica que caracterice el rendimiento (por ejemplo, el uso efectivo del ancho de banda) en la red subyacente.

En nuestro trabajo el concepto es utilizar estas nociones básicas para reconstruir la topología de una red de capa 3 *multivendor*, pero utilizando la automatización de tareas de red por medio de una librería que actúa además como capa de abstracción a la tecnología del fabricante que exista en la red.

2.4. Automatización de tareas de red con Python

La programabilidad de la red es una tendencia que se basa en métodos de *scripting* y lenguajes de programación estándar utilizados para controlar y supervisar los elementos de la red. Todas las nuevas implementaciones de automatización se basan en métodos de programación genéricos (Python, Java) e interfaces estándar (*Secure Shell SSH* o incluso RESTful Web Services) [44].

En esencia, la programabilidad y la automatización tienen como objetivo principal simplificar las tareas de configuración, gestión y funcionamiento de los equipos, la topología, los

servicios y la conectividad de la red [45].

Las bibliotecas con más uso para controlar los dispositivos de red de forma programática y con paquetes de código abierto son Paramiko y Netmiko, basados en Python. Tanto una como la otra utilizan la conexión SSH para obtener el control de los dispositivos. SSH es un protocolo de red criptográfico para operar servicios de red de forma segura a través de una red insegura [44].

Paramiko es una implementación en Python del protocolo SSHv2, que proporciona funcionalidad tanto de cliente como de servidor. Es una interfaz pura de Python en torno a los conceptos de red SSH y aprovecha una extensión de Python C para la criptografía de bajo nivel.

Netmiko es una librería multiproveedor, basada en Paramiko, que simplifica las conexiones a través de un amplio conjunto de proveedores de redes y plataformas. Tanto con una como con la otra, si necesitáramos extraer información acerca de la configuración de los equipos, tendríamos que reescribir el código de cada instrucción según el fabricante. Esto resulta incómodo y hace perder mucho tiempo a la hora de enfrentarse a un escenario multifabricante.

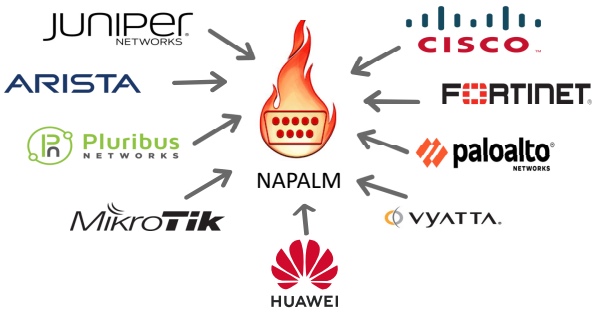


Figura 2.4: *Fabricantes a los que brinda soporte NAPALM.*

(Elaborado a partir de [46])

En nuestro trabajo utilizamos una biblioteca que se ha puesto de moda dentro de este tipo de desarrollos. Hablamos de NAPALM. Una herramienta que toma lo mejor de cada una de sus predecesoras utilizando el concepto de controlador genérico. NAPALM (*Network Automation and Programmability Abstraction Layer with Multivendor*) es una librería en

Python que implementa un conjunto de funciones para interactuar con dispositivos de diferentes proveedores utilizando una API unificada. Tener una API te permite elaborar tu código bajo esquemas universales, casi siempre de fácil comprensión y que resulta en ahorro en los tiempos de aprendizaje y desarrollo.

Los proveedores heterogéneos se integran mediante controladores, y NAPALM ofrece soporte para la mayoría de los proveedores importantes. Soporta varios métodos para conectarse a los dispositivos, manipular la configuración o recuperar datos, e integrarla con Ansible, Chef, Salt, etc. para configurar los elementos como dispositivos programables [47], [48], [44], [49].

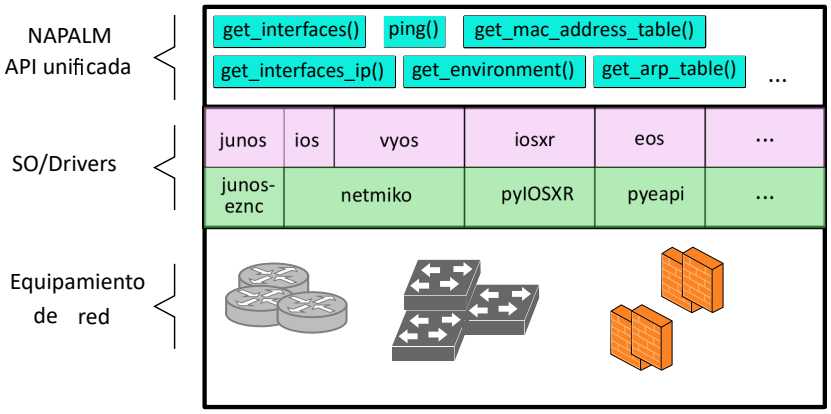


Figura 2.5: *Arquitectura de funcionamiento de NAPALM.*

(Elaborado a partir de [50])

También es posible utilizar YANG con NAPALM mediante una extensión. De esta manera se puede llegar a modelar los datos usados para crear los pedidos de la configuración del dispositivo o las peticiones los datos operativos por ejemplo del comando *show* o sus similares. Esta característica hace aún mas potente a la librería al incluir la posibilidad de que los datos tengan una formato estructurado en aplicaciones que así lo requieran.

Ahora, si utilizamos directamente NAPALM con los controladores de red basta conocer los métodos universales y qué nos devolverían en caso de aplicarlo a un equipo. Por citar algunos casos, si aplicamos *get_arp_table()* estaremos recibiendo información relativa, como se intuye, de la tabla ARP (*Address Resolution Protocol*).

```
{
    1: {
        "name": "default",
        "interfaces": ["GigabitEthernet0/0/1", "GigabitEthernet0/0/2"]
    },
    2: {
        "name": "vlan2",
        "interfaces": []
    }
}
```

(a) *get_vlans()*

```
[
    {
        'interface' : 'MgmtEth0/RSP0/CPU0/0',
        'mac'       : '5c:5e:ab:da:3c:f0',
        'ip'        : '2001:db8:1:1::1',
        'age'       : 1454496274.84,
        'state'     : 'REACH'
    },
    {
        'interface': 'MgmtEth0/RSP0/CPU0/0',
        'mac'       : '66:0e:94:96:e0:ff',
        'ip'        : '2001:db8:1:1::2',
        'age'       : 1435641582.49,
        'state'     : 'STALE'
    }
]
```

(b) *get_ipv6_neighbors_table()*

```
{
   'mircea': {
       'level': 15,
       'password': '$1$0P70xKPa$z46fewjo/10cBTckk6I/w/',
       'sshkeys': [
           'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC4pFn+shPwTb2yELO4L7NtQrKOJXNeC11je
       ]
   }
}
```

(c) *get_users()*

Figura 2.6: *Resultado de aplicar métodos de NAPALM.*

(Tomado de [51])

Por otra parte, ejecutar *get_bgp_config()* retorna la configuración del protocolo BGP (*Border Gateway Protocol*), *get_interfaces_counters()* las estadísticas de paquetes transmitidos y recibidos o *get_ipv6_neighbors_table()* una tabla con información referente al protocolo IP en su versión 6. Si hacemos *get_users()* nos devuelve datos acerca de los usuarios, *get_vlans()* de las redes virtuales de área local configuradas; por solo citar unos pocos ejemplos de todo lo que se puede llegar a hacer.

3 Capítulo 3: Implementación

3.1. Esquema general de la propuesta

Ha quedado claro que el objetivo central de nuestro trabajo es poder reconstruir la topología de red física de un operador para la cual no es restrictiva a un fabricante y/o variedad de los equipos. Para esto nos centraremos en lo que seria una red de transporte de alta capacidad, dejando fuera de nuestro análisis el segmento de la "ultima milla" o demás segmentaciones de más bajo nivel en la red. Con este concepto, proponemos un esquema como el de la figura 3.1 para solucionar el problema propuesto. El principio de funcionamiento radica en el uso de NAPALM [49] un *framework* de Python de relativa novedad que se utiliza para la automatización de tareas de red.

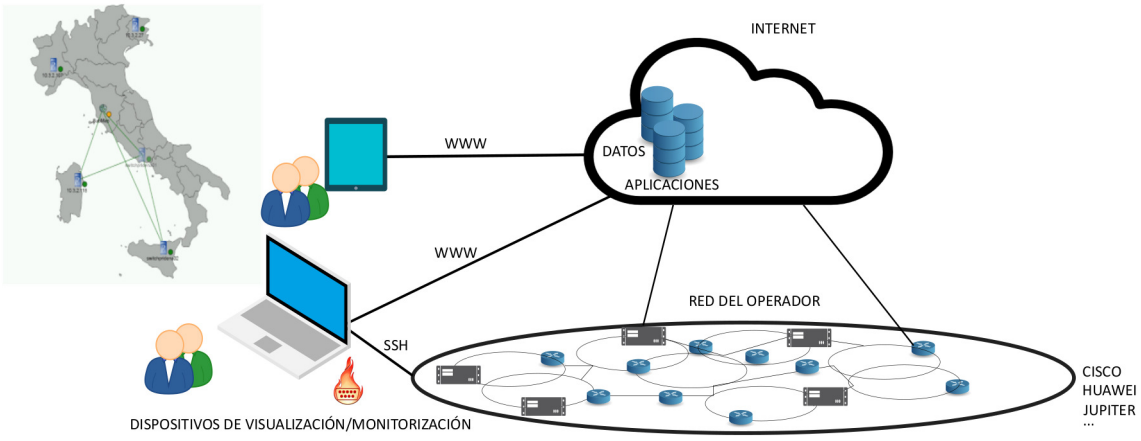


Figura 3.1: *Esquema general de la solución.*

Un caso de uso real y que sigue el esquema general de la figura 3.1 sería como el siguiente:

1. Un operador tiene una red como casi siempre, multifabricante. Este operador cuenta con más o menos condiciones, de un NOC (*Network Operations Center*) desde donde constantemente se supervisa la red, y desde el cual muchas veces es posible gestionar los equipos. Con esto asumimos que los mismos son accesibles remotamente desde este punto por medio de una red de gestión (otras formas de acceso a la red de gestión como la utilización de VPN está contemplado en este caso de uso).

2. Desde un punto del NOC nos conectamos a la red de gestión por medio de un ordenador. En este ordenador tendremos (ya sea gracias al almacenamiento en la nube u otras formas de intercambiar archivos) dos *scritps* y un archivo en formato Excel (extensión del paquete *Office*) que almacena la información de inventario de la red. Estos *scritps* se ejecutan en el ordenador luego de programarlos como tareas, desde las opciones de los distintos sistemas operativos (por ejemplo en *Windows*, *Windows Task Scheduler*). Mediante ellos se accede por SSH (*Secure Shell*) a los dispositivos.

3. La información recogida por los *scritps* anteriores se envían a una nube donde un servicio espera para organizar la información, por medio de una interfaz de programación de aplicaciones (más conocida como API) también alojada en el *cloud* y que trabaja junto a una base de datos de tipo SQL. Esta arquitectura soporta una aplicación web accesible desde Internet, mediante la cual es posible reconstruir la topología física de la red que se tiene en inventario y/o la red que se encuentra operativa en ese momento; llegándose a observar los cambios en la topología a causa del fallo de los equipos o de los enlaces entre los mismos, para que el personal encargado pueda devolver la red a su estado original (inventariado) en el que fue diseñada.

Este escenario real en el que interviene la infraestructura de un operador, por tratarse de un elemento crítico para su modelo de negocio, resulta imposible acceder a ella para realizar nuestro trabajo. Ni tan siquiera es posible conocer con claridad las configuraciones internas,

número IP (*Internet Protocol*) de los equipos, fabricantes específicos de cada segmento entre otras particularidades.

3.1.1. Solución particular para la reconstrucción de la topología de una red multifabricante

Por la razón expuesta en el final de la anterior sección de este capítulo es que se recurre a los software de simulación y/o entrenamiento que los diferentes fabricantes ponen a disposición de su comunidad. Estos por medio de la virtualización nos brindan una copia fiel de los equipos e incluso permiten conectar con redes reales. Los simuladores hacen posible a los desarrolladores replicar los modelos que esperan ver en el mundo real. Para tecnologías Juniper y Cisco es muy recomendable el GNS3 [52], WinBox para equipos Mikrotik [53], eNSP [54] para Huawei, entre otros. Así, con más o menos rigor los fabricantes ponen a disposición de los especialistas estas herramientas de trabajo.

Entonces la figura 3.1 adaptada a esta situación seria:

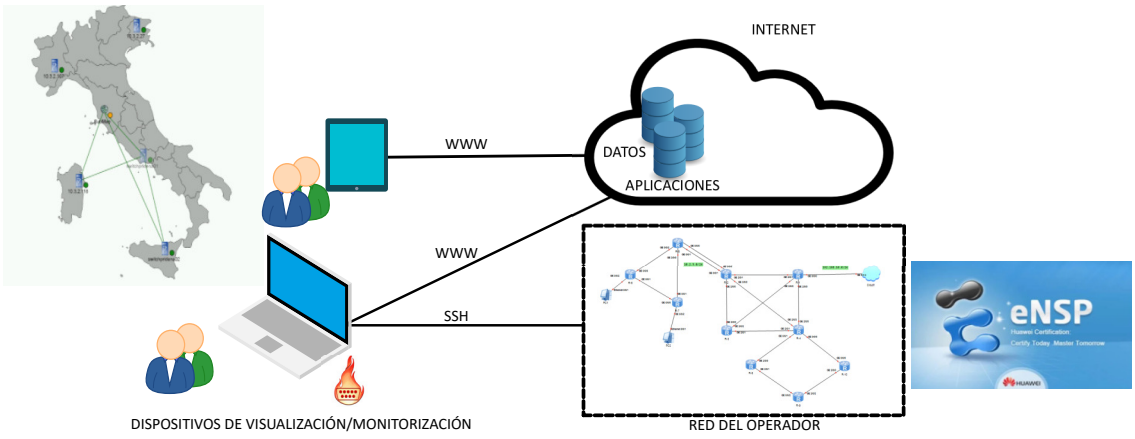


Figura 3.2: *Esquema de la solución particular.*

Hemos sustituido la red del operador real por una red simulada en eNSP. Esta red cumple con los requisitos mínimos de configuración de los servicios y protocolos para poder llevar

a cabo nuestros objetivos. Una red en funcionamiento tiene muchos más servicios y ajustes asociados, por ejemplo, a la teoría de tráfico pero que no forman parte del objeto de nuestro trabajo.

De esta manera podemos simular la red construida topológica y funcionalmente exacta, a la que se encuentra diseñada en el inventario, en un espacio que permite ser tomado como banco de pruebas sin afectar el rendimiento o los servicios a los clientes del operador de telecomunicaciones.

3.2. Desarrollo de la solución

A la hora de proponer una solución al problema de la reconstrucción de la topología de red multifabricante de un operador, se hace necesario hacer determinadas concesiones desde el punto de vista del escenario real y de las condiciones iniciales. Por ejemplo, a pesar de que existen regulaciones, que obligan a identificar la presencia de elementos radiantes, los operadores no tienen accesible al público en general los datos de sus inventarios, ni de como esta configurada su red, ni mucho menos de su modelo de negocio.

Por estas razones, en esta sección abordaremos las condiciones iniciales y desde las cuales partimos para llegar a nuestro resultado final. También describimos paso a paso la herramienta desarrollada.

3.2.1. Inventario de la red

La gestión eficiente de los recursos y en especial de los dispositivos, necesita de una especial importancia, de lo contrario la dinámica general de la empresa puede estar comprometida.

Si fuera necesario, por citar un ejemplo, una migración de un equipo o un cambio de

enrutamiento, es extremadamente complicado ejecutarla sin visión clara del ambiente con el cual estamos lidiando. Cualquier acción del equipo de TI se vuelve limitada y arriesgada sin las informaciones exactas de cuales son los elementos, donde están localizados, que relación existen entre ellos etc.

La idea de un inventario de red fue creada con el objetivo de resolver esta cuestión. Dar solución a un problema se vuelve mucho más ágil si sus profesionales tienen acceso inmediato a un documento que indica todo aquello que se tiene: servidores, computadores, dispositivos móviles, *switches*, *routers*, cables entre otros.

El inventario de red recoge cada uno de esos elementos y los relaciona con sus informaciones técnicas de forma detallada. Aporta confiabilidad ya que se supone que lo registrado en él es realmente con lo que se cuenta. En este documento es un punto importante registrar la ubicación de cada elemento.

Para nuestro trabajo, partimos de un documento en Excel el cual recoge todos los equipos que tiene (en el supuesto que hemos planteado), el operador en su red de transporte de alta capacidad. A este lo hemos llamado *inventario_red.XLSX*. Agregar que para el funcionamiento de la herramienta, es necesario que este archivo y los dos *scripts* que comentaremos más adelante estén en el mismo directorio.

Aclarar que en nuestro inventario concretamente lo que se guardan son las interfaces de red de los dispositivos, y que dado los equipos se componen de las interfaces es equivalente decir que en nuestro archivo de inventario se guardan los equipos que conforman la red.

El documento está compuesto de nueve columnas, las cuales siguen este orden:

- VENDOR: Hace referencia al nombre del fabricante y/o vendedor del elemento de red (para nuestro caso particular, Huawei).
- HOSTNAME: Este campo se refiere al nombre que tiene el equipo (por ejemplo R-1, R-2, ...).

- UBICATION: Se introduce para poder geolocalizar al equipo. Cuando una de las filas de nuestro inventario se refiera a la interfaz de Loopback0 de un equipo, esta lleva en el campo en cuestión la tupla [latitud, longitud] donde se encuentra el dispositivo. Para el resto de las interfaces este campo es nulo.

- INTERFACE: Aquí guardamos el nombre de la interfaz (LoopBack0, GigabitEthernet0/0/0, GigabitEthernet0/0/1, etc).

- IPV4: Campo destinado al número IP versión 4 que tiene la interfaz. Para una interfaz de Loopback0 el IPv4 será del tipo 10.10.10.X (con máscara 32), siendo X el número del equipo. Las demás interfaces estarán dentro de una red del tipo X.X.X.252 (máscara 30) quedando la IP X.X.X.253 para la interfaz de un extremo de la conexión entre dos equipos y X.X.X.254 para la otra.

- MASK: Guardamos la máscara de red de la interfaz en formato decimal (por ejemplo 32, 30, 24).

- IS_UP: Con un *False* la interfaz no esta funcionando y con un *True* sí. Por defecto en el inventario partimos que están en estado *False*.

- IS_ENABLE: Con un *False* la interfaz no esta habilitada y con un *True* sí. Por defecto en el inventario partimos que están en estado *False*.

- MAC: En este campo almacenamos la MAC, identificador que corresponde de forma única a una tarjeta de red física. Para las interfaces de Loopback0 este valor es nulo al no ser una interfaz física real.

En la figura 3.3 se muestra un fragmento del archivo considerado de inventario. La disposición de las columnas es primordial para que funcione correctamente nuestra solución. La interfaz con IPv4 192.168.10.3 corresponde con la interfaz por la cual nos vamos a conectar con la red en el simulador, es por eso que decidimos diferenciarla de las demás.

A	B	C	D	E	F	G	H	I
VENDOR	HOSTNAME	UBICATION	INTERFACE	IPV4	MASK	IS_UP	IS_ENABLE	MAC
Huawei	R-1	41.5193, 2.0776	LoopBack0	10.10.10.1	32	False	False	
Huawei	R-1		GigabitEthernet0/0/0	192.168.10.3	24	False	False	00:E0:FC:F0:49:13
Huawei	R-1		GigabitEthernet2/0/0	10.1.4.253	30	False	False	00:E0:FC:F3:49:13
Huawei	R-1		GigabitEthernet2/0/1	10.1.2.253	30	False	False	00:E0:FC:F3:49:14
Huawei	R-1		GigabitEthernet2/0/2	10.1.3.253	30	False	False	00:E0:FC:F3:49:15
Huawei	R-2	41.8024, -1.0318	LoopBack0	10.10.10.2	32	False	False	
Huawei	R-2		GigabitEthernet0/0/0	10.5.2.254	30	False	False	00:E0:FC:9A:4D:28
Huawei	R-2		GigabitEthernet0/0/1	10.2.5.254	30	False	False	00:E0:FC:9A:4D:29
Huawei	R-2		GigabitEthernet2/0/0	10.2.3.254	30	False	False	00:E0:FC:9D:4D:28
Huawei	R-2		GigabitEthernet2/0/1	10.1.2.254	30	False	False	00:E0:FC:9D:4D:29
Huawei	R-2		GigabitEthernet2/0/2	10.2.4.254	30	False	False	00:E0:FC:9D:4D:2A
Huawei	R-3	40.5437, -3.6970	LoopBack0	10.10.10.3	32	False	False	
Huawei	R-3		GigabitEthernet2/0/0	10.2.3.253	30	False	False	00:E0:FC:FA:76:DB
Huawei	R-3		GigabitEthernet2/0/1	10.3.4.253	30	False	False	00:E0:FC:FA:76:DC
Huawei	R-3		GigabitEthernet2/0/2	10.1.3.254	30	False	False	00:E0:FC:FA:76:DD
Huawei	R-4	39.6039, -0.3524	LoopBack0	10.10.10.4	32	False	False	
Huawei	R-4		GigabitEthernet0/0/0	10.4.10.254	30	False	False	00:E0:FC:2D:10:C6
Huawei	R-4		GigabitEthernet0/0/1	10.4.8.254	30	False	False	00:E0:FC:2D:10:C7
Huawei	R-4		GigabitEthernet2/0/0	10.1.4.254	30	False	False	00:E0:FC:30:10:C6
Huawei	R-4		GigabitEthernet2/0/1	10.3.4.254	30	False	False	00:E0:FC:30:10:C7
Huawei	R-4		GigabitEthernet2/0/3	10.2.4.253	30	False	False	00:E0:FC:30:10:C9

Figura 3.3: *Vista preliminar del archivo de inventario.*

3.2.2. Red desplegada en eNSP

Como ya hemos dicho era imposible poder acceder a una red real de un operador para testear el despliegue de la aplicación. Es por eso que nos decidimos por una plataforma de simulación de red gráfica, ampliable y gratuita desarrollada por Huawei, eNSP (*Enterprise Network Simulation Platform*).

Con eNSP se puede simular redes de gran tamaño, realizar pruebas, aprender las tecnologías y protocolos sin usar dispositivos reales.

En la figura 3.4 tenemos la red que hemos configurado en la plataforma de Huawei. A la derecha en la imagen, situamos lo que se corresponde en la arquitectura general a la conexión desde el NOC a la red de gestión. A la izquierda, dos *PCs* que simulan un equipo terminal para esta red de transporte, por ejemplo una *gNB*[1], un cliente importante u otro equipo de menor

[1]Nombre con el cual se conoce la estación base de redes 5G.

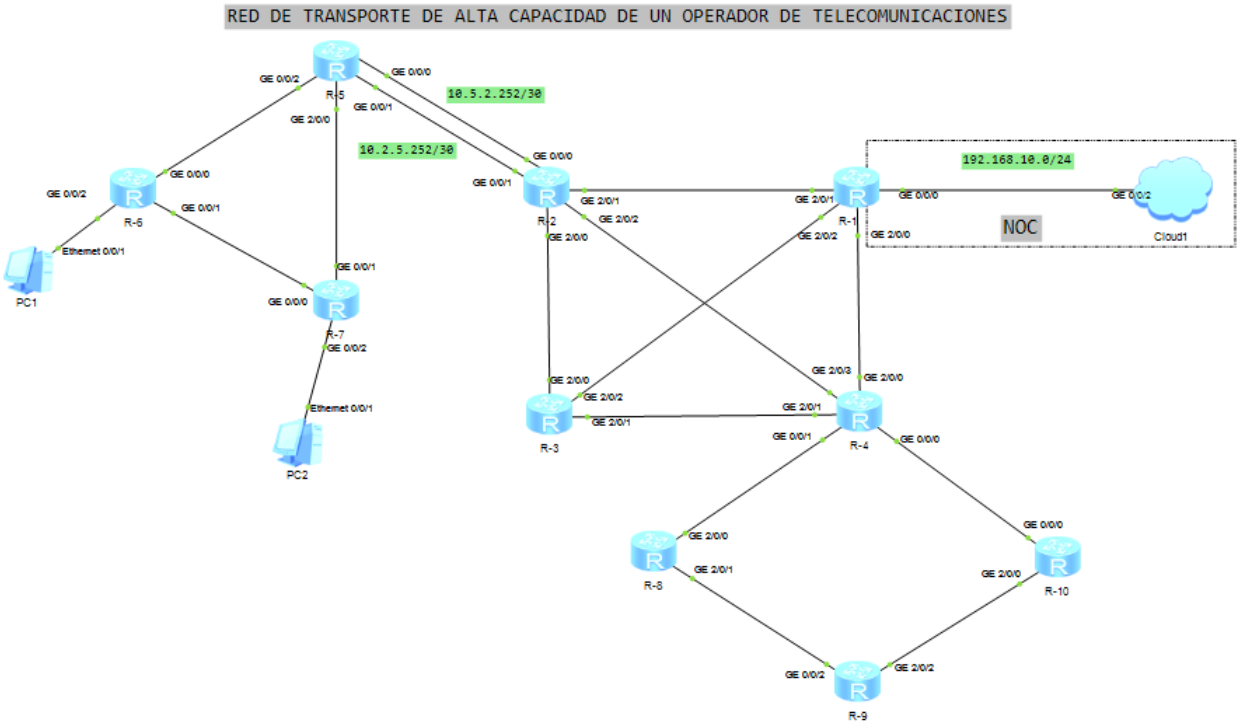


Figura 3.4: *Red desplegada en eNSP.*

jerarquía en la red.

Los equipos en la topología simulada son alcanzables entre ellos, es decir desde cualquiera de los dispositivos (incluyendo las *PCs*) se puede obtener un *ping* [2] satisfactorio a las interfaces de Loopback0 de los *routers* o hacia las *PCs*. Lo hemos conseguido, gracias a propagar estas rutas mediante OSPF (*Open Shortest Path First*).

OSPF es un protocolo de ruteo interno basado en el estado del enlace lo cual es una ventaja con respecto a los demás protocolos de su tipo, ya que las rutas se crean conociendo el estado completo de la red. De esta forma se hace un enrutamiento mucho más eficiente, eligiendo el camino más corto primero. El protocolo utiliza el algoritmo de Dijkstra para determinar la mejor ruta a seguir [55]. A partir de una actualización, un *router* crea una base de datos topológica que posibilita calcular la accesibilidad a las redes gracias al cálculo de un árbol de la topología de la que el *router* es la raíz [55].

[2] Comando de diagnóstico que comprueba el estado de la comunicación en redes IP.

La métrica para el cálculo en este algoritmo del camino más corto es el costo. Cada interfaz dentro de un área OSPF tiene un valor de costo. La fórmula para calcularlo es $costo = 100M/BW$, siendo BW el ancho de banda de la interfaz y 100 Mbit/s el valor de ancho de banda de referencia en OSPF [55]. Por tanto, el costo de una ruta es la suma de los costos de las interfaces desde el *router* fuente hasta el destino.

En nuestro caso, hemos planificado un funcionamiento mínimo de la red para que exista comunicación. Es decir, la red es capaz de comunicar un extremo (las *PC* que hemos puesto de prueba) con lo que llamamos NOC. Desde el punto de vista del protocolo OSPF configuramos una única área en la cual cada *router* de la topología propaga las redes que conoce, que no son más que a la que pertenece su interfaz de *Loopback0* y las demás redes de sus otras interfaces. Un ejemplo para uno de los dispositivos se muestra en la figura.

```
#
interface LoopBack0
 description 43.4841, -8.3874
 ip address 10.10.10.6 255.255.255.255
#
ospf 1 router-id 10.10.10.6
 area 0.0.0.0
  network 10.5.6.252 0.0.0.3
  network 10.6.7.252 0.0.0.3
  network 10.10.10.6 0.0.0.0
  network 100.1.1.0 0.0.0.255
#
```

Figura 3.5: *Configuración del protocolo OSPF.*

En la imagen se aprecia el resultado de aplicar en la consola del *router* R-6 el comando *display current-configuration* para que nos devuelva la configuración que tiene el equipo. Dentro de la información que se muestra, capturamos la que corresponde a la configuración del protocolo OSPF y a la interfaz *Loopback0*, para ilustrar lo que explicamos anteriormente con respecto al atributo descripción, campo que para hacer que funcione nuestra solución, debe guardarse en él por el personal calificado a la hora del despliegue, la tupla [latitud, longitud] de donde está ubicado el equipo.

El modelo de *router* utilizado en la topología desplegada en eNSP es el AR2240. Este equipo es considerado de uso empresarial y en la vida real no es el que se utiliza para las redes de transporte de alta capacidad de un operador como si lo puede ser el NE40E, NE5000E o

NE9000. Ahora, si con estos dispositivos en eNSP, quisiéramos simular una red de la magnitud que tenemos (unos 10 equipos) la práctica demostró que el coste computacional es elevado, incluso insuficiente para un ordenador como con el que contamos y que sus prestaciones no son despreciables (Intel Core i7-3537U 2.500GHz, 8 GB de RAM). Esta decisión al final no condiciona la realidad del escenario que planeamos ya que en cualquiera de uno u otro equipo de los mencionados las configuraciones que se realizan ocurren de la misma manera, con los mismos comandos y secuencias.

3.2.3. Servicio web en la nube

En la nube desplegamos un servicio web de tipo REST (*Representational State Transfer*) más concretamente una API REST. Esto no es más que "cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (*Simple Object Access Protocol*), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST" [56].

REST separa totalmente la interfaz de usuario del servidor y del almacenamiento de datos. Por ejemplo, "mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente" [56].

La API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. "Con una API REST se pueden tener servidores PHP, Java, Node.js o Python. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON" [56].

Utilizamos para el *host* del servicio a Heroku, que "es una plataforma como servicio (PaaS)

que permite a los desarrolladores crear, ejecutar y operar aplicaciones completamente en la nube" [57]. Esta tiene un plan gratuito y fue el que elegimos. Para la base de datos utilizamos mySQL y Node.js para el desarrollo de la API.

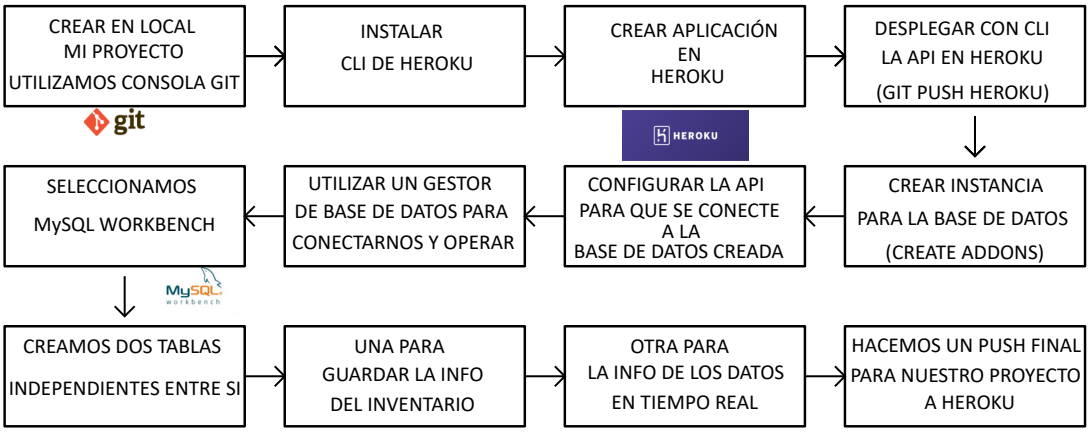


Figura 3.6: *Secuencia del desarrollo del servicio web.*

El código que compone a la API, básicamente establece el puerto por el cual estará escuchando peticiones, la conexión con la base de datos mediante usuario y contraseña y lo que se conoce como CRUD. Este acrónimo hace referencia a los métodos que implementa la API sobre la base de datos: la C de *Create*, R de *Read*, U de *Update* y D de *Delete*; métodos que se corresponden con las operaciones de GET (obtener), POST (añadir, publicar), PUT (publicar) y DELETE (eliminar).

En la figura 3.7 se presenta un fragmento del código de la API. En primera instancia tenemos la conexión a la base de datos por medio de usuario y contraseña que se han protegido por privacidad. En un segundo momento podemos apreciar un método de tipo GET con su correspondiente consulta SQL, en particular este método devuelve todo lo referente a la tabla de inventario.

Al terminar este proceso, quedarían disponibles las rutas para poder ejecutar acciones sobre la base de datos a través de la API. En `https://huawei-app.herokuapp.com` el servicio queda disponible. Las rutas serán usadas más adelante tanto para guardar información como para proveer de la misma a una aplicación web.

```
// MySql
const connection = mysql.createPool({
  host: 'us-cdbr-east-03.cleardb.com',
  user: '[redacted]',
  password: '[redacted]',
  database: 'heroku_d4466ec00137453'
});

app.get('/interfaces_inv/all', (req, res) => {
  const sql = 'SELECT * FROM interfaces_inv';

  connection.query(sql, (error, results) => {
    if (error) throw error;
    if (results.length > 0) {
      res.json(results);
    } else {
      res.send('Empty Inventory');
    }
  });
});
```

Figura 3.7: *Fragmento de código de la API.*

3.2.4. Recolección de los datos

Esta sección la dedicaremos a detallar los *scripts* que desarrollamos en Python para la recolección de los datos del inventario y de la red en tiempo real. Partimos de la hipótesis de que si tenemos esta información, será posible posteriormente crear una aplicación que nos muestre gráficamente la topología de la red inventariada y la que funciona en tiempo real para compararlas y detectar incidencias.

El objetivo es programar mediante la aplicación de tareas que normalmente traen los sistemas operativos (por ejemplo *Windows Task Scheduler*) para que estos *scripts* se ejecuten tantas veces en el día como se necesite o hacerlo de manera manual cada vez que se quiera obtener la información actualizada.

Recolección de los datos del inventario

Para recuperar la información del archivo de inventario fue necesario desarrollar un *script* que cumpla con esta función, lo hemos llamado *load_inventory.py*. Básicamente la secuencia que utilizamos fue en primer lugar hacer un borrado de la tabla inventario en la base de datos para asegurarnos que los datos que quedarán almacenados al final del proceso son los más actualizados. En segundo lugar y utilizando la biblioteca *xlrd* para manejar archivos Excel, recuperamos la información de una fila a la vez y la enviamos a la base de datos por medio de una solicitud POST. Este proceso se repite hasta recorrer todo el inventario.

```
10  # Give the location of the file
11    loc = "inventario_red.XLSX"
12
13  # To open Workbook
14    wb = xlrd.open_workbook(loc)
15    sheet = wb.sheet_by_index(0)
16
17    url = 'https://huawei-app.herokuapp.com/[redacted]'
18
19    for i in range(1,sheet.nrows):
20        myobj = {'vendor': sheet.cell_value(i, 0),
21                 'hostname': sheet.cell_value(i, 1),
22                 'ubication': sheet.cell_value(i, 2),
23                 'interface': sheet.cell_value(i, 3),
24                 'ipv4': sheet.cell_value(i, 4),
25                 'mask': sheet.cell_value(i, 5),
26                 'is_up': sheet.cell_value(i, 6),
27                 'is_enable': sheet.cell_value(i, 7),
28                 'mac': sheet.cell_value(i, 8)
29                 }
30        x = requests.post(url, json = myobj)
31        print(x.text)
```

Figura 3.8: *Fragmento de código del script load_inventory.*

Es necesario tener instalado Python 3 en el ordenador donde se ejecuta para ejercer de intérprete. Además debemos colocar en el mismo directorio el archivo Excel y este *script*.

Recolección de los datos de la red en tiempo real

El objetivo de este otro *script* que hemos llamado *load_realtime.py* es recuperar el estado actual de la red. La siguiente figura ilustra la secuencia que seguimos en el *script* para conseguir el objetivo deseado.

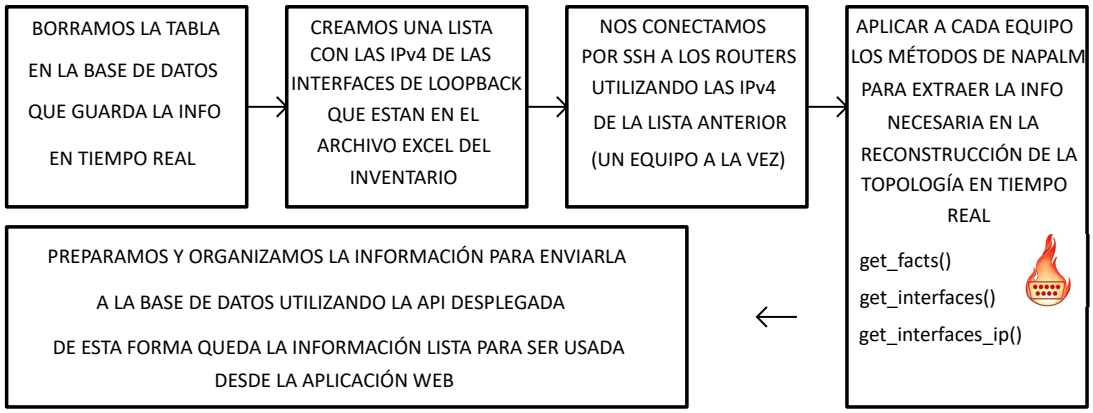


Figura 3.9: *Secuencia de desarrollo de la recolección del estado actual de la red.*

Para ejecutar el programa utilizamos la consola de Ubuntu que trae incorporada el Sistema Operativo Windows 10 de nuestro ordenador. Es recomendable utilizar una consola para poder observar los mensajes de estado que hemos incorporado al código. Estos mensajes nos sirven para ir conociendo si la secuencia de acciones se van produciendo correctamente.

Los métodos de NAPALM que utilizamos para extraer la información fueron:

- get_facts(): devuelve características generales del equipo como por ejemplo nombre (*hostname*), fabricante (*vendor*), modelo (*model*), tiempo de funcionamiento (*uptime*), numero de serie (*serial_number*), entre otros.

- get_interfaces(): esta función permite obtener información referente a la interfaz de red pero que no este relacionado con el protocolo IP. Por ejemplo saber si está funcionando la interfaz (*is_up*), si está habilitada (*is_enabled*), la descripción (*description*), la MAC (*mac_address*), velocidad (*speed*), MTU [3](*Maximum Transfer Unit*).

[3]Tamaño en bytes de la unidad de datos más grande que puede enviarse con un protocolo de comunicaciones.

- get_interfaces_ip(): con este método recuperamos lo asociado al protocolo IP de cada interfaz (IP y máscara).

En la figura 3.10 presentamos una parte del código de este programa. El primer ciclo en el fragmento es el encargado de formar la terna compuesta por el IP del *router* al cual nos vamos a conectar, el identificador del sistema operativo del *router* (al ser Huawei es VRP) y el tipo de dispositivo (*router* o *switch*). Con esto estamos preparados para en el segundo ciclo poder establecer la conexión por medio de usuario y contraseña.

```
25      for i in range(1,sheet.nrows):
26          if sheet.cell_value(i, 3) == "LoopBack0":
27              aux = sheet.cell_value(i, 4)
28              device_list.append([aux,"vrp", "router"])
29
30      network_devices = []
31      for device in device_list:
32              network_devices.append(
33                                  driver_vrp(
34                                  hostname = device[0],
35                                  username = "huawei",
36                                  password = "huawei"))
```

Figura 3.10: *Fragmento de código del script load_realtime (1).*

Con la figura 3.11 podemos observar de que manera se accede a cada equipo y como preparamos una excepción para el caso de que el dispositivo no exista o no esta funcionando. Posteriormente si es accesible, empezamos a recopilar los primeros datos que son de nuestro interés.

Posteriormente estamos en condiciones de recopilar los demás datos referentes a la interfaz y que completarían todo lo necesario que nos planteamos para reconstruir la topología de red en tiempo real. Alguna información hay que manipular por medio de instrucciones como *split* para poder extraerla de forma organizada. Muestra de ello en la figura 3.12.

```
42      for device in network_devices:
43       flag = 0
44       try:
45          print("Connecting to {} ...".format(device.hostname))
46          device.open()
47       except napalm.base.exceptions.ConnectionException:
48          print("{} Inaccessible host ".format(device.hostname))
49          flag = 1
50
51       if flag == 0:
52          print("Getting device facts")
53          device_facts = device.get_facts()
54
55          devices_table.append([device_facts["hostname"],
56                                device_facts["vendor"],
57                                device_facts["model"],
58                                device_facts["uptime"],
59                                device_facts["serial_number"]
60                                ])
```

Figura 3.11: *Fragmento de código del script load_realtime (2).*

```
64          device_interfaces_ip = device.get_interfaces_ip()
65          device_interface = device.get_interfaces()
66          for interface in device_interfaces_ip:
67              is_up = device_interface[interface]['is_up']
68              is_enabled = device_interface[interface]['is_enabled']
69              mac_address = device_interface[interface]['mac_address']
70              var = device_interfaces_ip[interface]['ipv4']
71              speed = device_interface[interface]['speed']
72              if interface == "LoopBack0":
73                  ubication = device_interface[interface]['description']
74              else:
75                  ubication = ""
76              aux = str(var)
77              aux = aux.split("'")
78              ip = aux [1]
79              mask = aux [4]
80              mask = mask[2:4]
```

Figura 3.12: *Fragmento de código del script load_realtime (3).*

Con toda la información recogida, solo queda enviarla a la base de datos, a la tabla *interfaces_realtime* por medio de la API, para que así este lista para mostrarla desde la aplicación web. Ilustramos en figura 3.13.

```
87                  myobj = {'vendor': vendor,
88                           'hostname': hostname,
89                           'model': model,
90                           'ubication': ubication,
91                           'uptime': uptime,
92                           'serial_number': serial_number,
93                           'interface': interface,
94                           'ipv4': ip,
95                           'mask': mask,
96                           'is_up': is_up,
97                           'is_enable': is_enabled,
98                           'mac': mac_address,
99                           'speed': speed
00                           }
01                  x = requests.post(url, json = myobj)
```

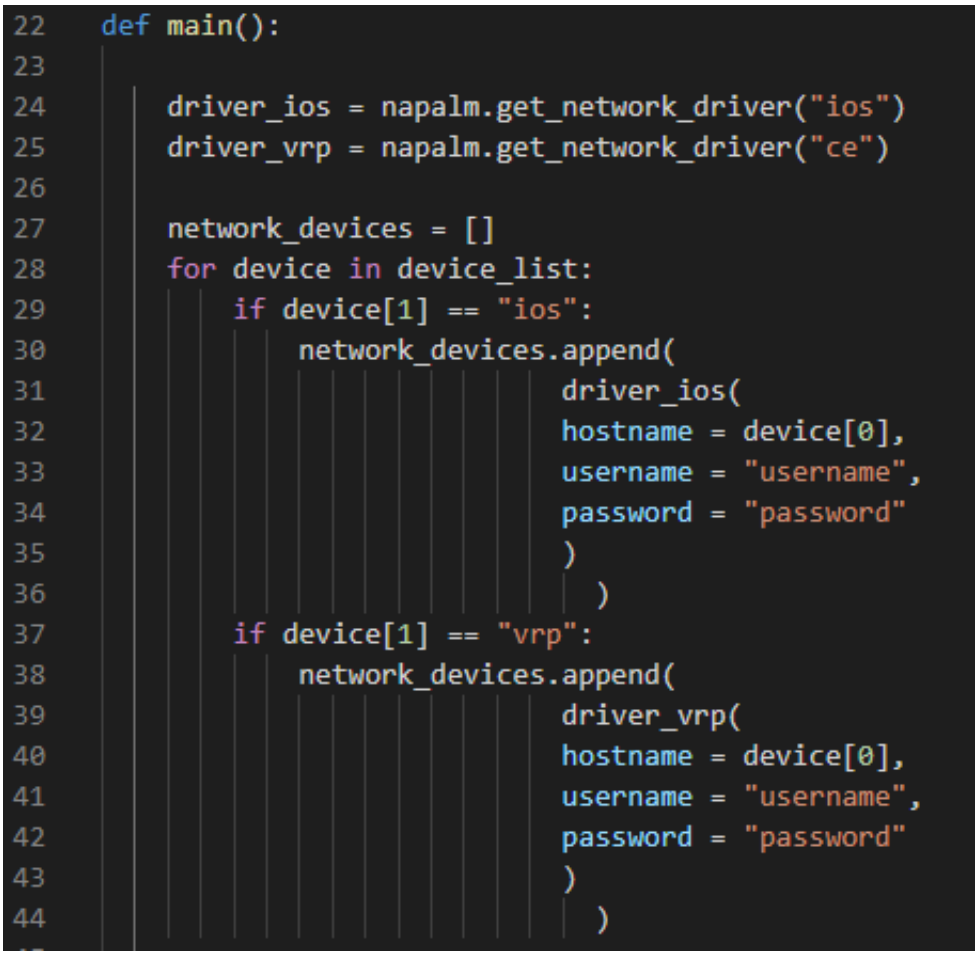
Figura 3.13: *Fragmento de código del script load_realtime (4).*

Colectar datos de múltiples fabricantes

Como hemos venido afirmando en nuestro trabajo es posible usando Napalm, incluso con un ciclo de instrucciones casi idénticas a las mostradas hasta ahora, poder colectar los datos de un escenario con múltiples fabricantes. Napalm da soporte a Juniper, Cisco, Mikrotik, Fortinet por solo citar los más importantes; y la comunidad de desarrolladores da soporte con NAPALM a Huawei por medio de un directorio en GitHub [4].

La figura 3.14 muestra el procedimiento a seguir para, por ejemplo, recolectar los datos de una red de un operador que sus equipos sean de Cisco y de Huawei.

[4]Repositorio en la nube para alojar proyectos utilizando el sistema de control de versiones Git



```
def main():

    driver_ios = napalm.get_network_driver("ios")
    driver_vrp = napalm.get_network_driver("ce")

    network_devices = []
    for device in device_list:
        if device[1] == "ios":
            network_devices.append(
                                driver_ios(
                                hostname = device[0],
                                username = "username",
                                password = "password"
                                )
                                  )
        if device[1] == "vrp":
            network_devices.append(
                                driver_vrp(
                                hostname = device[0],
                                username = "username",
                                password = "password"
                                )
                                  )
```

Figura 3.14: *Colecta de datos de múltiples fabricantes.*

Como se puede apreciar resulta sencillo. Solo queda indicar que tipo de *driver* utiliza el equipo. Si el equipo es Cisco utiliza el sistema operativo IOS y si es Huawei utiliza VRP. Luego los métodos de Napalm que aplicamos en un primer momento solo para el caso de Huawei son igual de funcional para Cisco, ya que usando Napalm tenemos una capa de abstracción que universaliza el trabajo y nos permite “hablar con todos en un mismo idioma”.

Ahora, esto no fue posible implementarlo para nuestro trabajo ya que hacen falta más recursos computacionales para poder simular por separado y para cada plataforma, una red con las dimensiones como las de este caso.

3.2.5. Aplicación web

Desarrollar una aplicación web era necesario para nuestro proyecto. Esta es la herramienta gráfica mediante la cual damos solución al problema de reconstruir la topología física de la red de un operador. Como no puede ser de otra manera, tenerla en la web permite acceder a ella desde cualquier terminal con acceso a Internet. Nos propusimos que la aplicación debía ser por sobre todo potente visualmente, en la cual ganara visibilidad la topología que íbamos a reconstruir.

Siguiendo las tendencia que marca el desarrollo de aplicaciones para Internet en los últimos tiempos, utilizamos Angular. Este es un *framework* de JavaScript potente de código abierto, mantenido por Google. Angular incluye una serie de módulos habituales en el desarrollo de proyectos web que no tienes que desarrollar desde cero y que permiten organizar un proyecto de una manera óptima [58].

Angular usa TypeScript para la programación. Este "es una extensión del lenguaje JavaScript, que agrega características importantes, como el tipado de datos o los decoradores. La compilación de este a Javascript es compatible con los navegadores y se realiza por medio de las herramientas del Angular CLI (*Command-Line Interface*), por lo que no agrega ninguna dificultad en el flujo de desarrollo" [58].

Para el trabajo con los mapas, elegimos Leaflet que es también un *framework* de JavaScript de código abierto, líder para el trabajo con mapas interactivos. Funciona de manera eficiente en todas las principales plataformas, se puede ampliar con muchos complementos, tiene una API fácil de usar y bien documentada, y un código fuente simple y legible [59]. La fuente de los mapas es *OpenStreetMap*.

La aplicación web que la hemos llamado Napalm-App consumirá los datos del servicio desplegado anteriormente. Esta cuenta con dos componentes fundamentales: *tabs* y *map*, donde el primero actúa como padre del segundo. En *tabs* tenemos la lógica para cargar el mapa con la topología en inventario o el mapa con la topología en tiempo real. El componente hijo *map*

envía actualizaciones al padre por medio de la clase *EventEmitter*. Desde *map* implementamos la función *handlerRouters* en el archivo *map.component.ts*, que será el encargado de emitir los mensajes al padre.

La función *handlerRouters* (Figura 3.15) separa en dos arreglos las interfaces de los *routers*. Un arreglo guarda las interfaces de LoopBack0 y el otro las interfaces físicas que se conectan entre si. Esto sucede cada vez que desde el *tabs.component.ts* se ejecute el método *selectedTabChange* que a su ves llama a *onChange* para tiempo real o red inventariada, según seleccionemos desde la aplicación. El emisor de eventos *handlerRouters* termina invocando a *setMarkers* (Figura 3.16) para geolocalizar los equipos y a *setConnections* para establecer la conexión o línea de unión entre los equipos.

```
48  handlerRouters(event: { type: string, routers: RouterInterface[] }): void {
49    const type = event.type;
50    const routers = event.routers;
51    console.log(routers);
52    let interfacesToConnect: RouterInterface[];
53    let interfacesLoopBack: RouterInterface[];
54    if (type === 'realtime') {
55      interfacesToConnect = routers.filter((router: RouterInterface) =>
56        (router.mask === '30' && router.is_enable && router.is_up));
57      interfacesLoopBack = routers.filter((router: RouterInterface) =>
58        (router.interface === 'LoopBack0' && router.is_enable && router.is_up));
59    } else {
60      interfacesToConnect = routers.filter((router: RouterInterface) => router.mask === '30');
61      interfacesLoopBack = routers.filter((router: RouterInterface) => router.interface === 'LoopBack0');
62    }
63    this.setMarkers(interfacesLoopBack, routers, type, interfacesToConnect);
64    this.setConnections(interfacesToConnect, interfacesLoopBack);
65  }
```

Figura 3.15: *Función handlerRouters en map.component.ts.*

```
79          const coordinates = router.ubication.split(',');
80          const mark = L.marker([+coordinates[0], +coordinates[1]], this.icon);
```

Figura 3.16: *Geolocalizando los routers con Leaflet.*

Para trazar las líneas que unen las interfaces físicas entre los *routers* con *setConnections*, establecimos los siguientes pasos:

- Red en tiempo real: De todas las interfaces de la tabla *interfaces_realtime* de la base de datos filtramos por las que tengan las máscara igual a 30 y el campo is_up y is_enable

tengan el valor 1. Luego se toma una interfaz, por ejemplo, GigabitEthernet0/0/1 de R-6 y si is_up e is_enable son 1 para ella, ahora su IPv4 es 10.6.7.253, y solo habrá dentro de todas las demás interfaces de la tabla una que tenga el valor de IPv4 10.6.7.X que será la IPv4 10.6.7.254 y con la cual se enlaza por medio de una línea simulando un enlace físico. Esto se va repitiendo dentro de un ciclo hasta que se tengan los pares a conectar. Luego con la función *polyline* de Leaflet dibujamos la conexión entre los dos puntos que ya anteriormente teníamos geolocalizados.

- Red inventariada: De todas las interfaces de la tabla de la base de datos *interfaces_inv* filtramos por las que tengan las máscara igual a 30, no harían falta los campos is_up y is_enable, porque partimos del supuesto que estos valen 0. Luego se toma una interfaz, por ejemplo, GigabitEthernet0/0/1 de R-6, de ella su IPv4 10.6.7.253. Ahora, solo habrá dentro de todas las demás interfaces de la tabla una que tenga el valor de IPv4 10.6.7.X que será la IPv4 10.6.7.254 y con la cual se enlaza por medio de una línea simulando un enlace físico. Esto se va repitiendo dentro de un ciclo hasta que se tengan los pares a conectar. Luego con la función *polyline* de Leaflet dibujamos la conexión entre los dos puntos que ya anteriormente teníamos geolocalizados.

Para garantizar un mínimo de seguridad, la plataforma cuenta con una sesión de *login*. Está publicada para acceder a la misma en `https://napalm-app.herokuapp.com/login`.

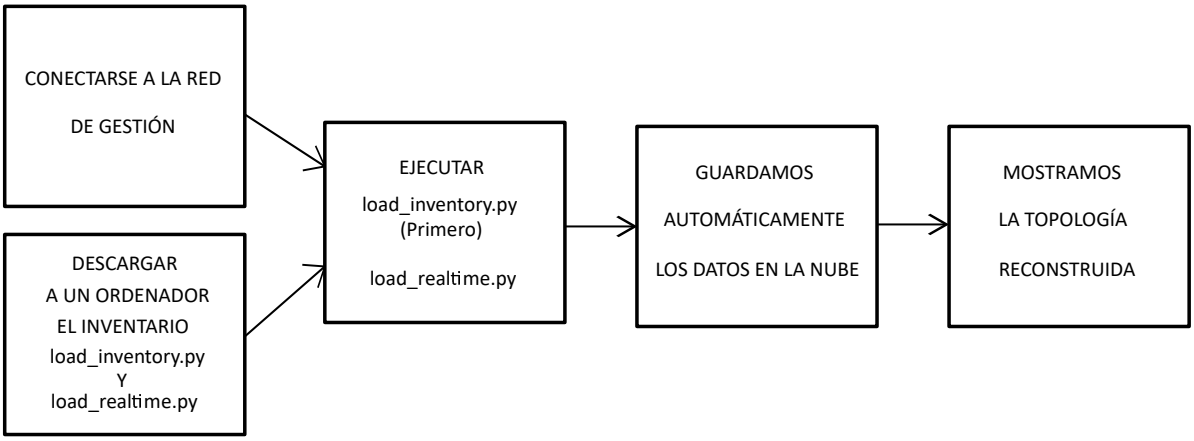


Figura 3.17: *Flujo final de trabajo.*

4 Capítulo 4: Experimentación

Llegado a este punto, estamos en condiciones de realizar un proceso de experimentación y pruebas a nuestra aplicación. De esta manera, comprobaremos que hemos dado solución a nuestro objetivo de reconstruir la topología de red de un operador de telecomunicaciones a partir de la configuración de sus elementos incluso en un escenario multifabricante.

4.1. Condiciones iniciales

Para nuestra solución ingenieril hizo falta ponernos un punto de partida en cuanto a las condiciones necesarias para que la aplicación funcione. Esto fue necesario para recrear lo que sería un escenario real desde un ordenador.

Primeramente necesitamos tener instalado el intérprete de nuestro código que no es más que Python 3. Además contamos con un archivo Excel el cual recoge el inventario de red según explicamos con anterioridad. Este archivo sería la guía de lo que en un primer momento se proyectó y diseñó para desplegar en campo. Por lo cual, aceptamos este inventario como el estado de la red deseado y punto de partida de comparación con el de tiempo real.

Al unísono, descargamos al ordenador los *scripts load_inventory.py* y *load_realtime.py* desde la fuente de almacenamiento donde este. Ambos deben colocarse en el mismo directorio del archivo de inventario.

Por ultimo tenemos la red que se ha desplegado siguiendo las directrices del inventario. Como se ha explicado la desplegamos en eNSP de Huawei, por no poder acceder a la red de un operador real de telecomunicaciones.

4.2. Pruebas de funcionamiento

Antes de comenzar con la secuencia de pruebas de funcionamiento comprobamos que los equipos en la topología en eNSP estén funcionando. En la figura 4.1 tenemos la misma red de la figura 3.4 configurada según el inventario, solo que la concepción del problema se reduce a sustituir lo que conocíamos por NOC por nuestro ordenador. En esta figura podemos observar como delante del nombre de cada interfaz, un pequeño punto verde nos indica que las interfaces están funcionando.

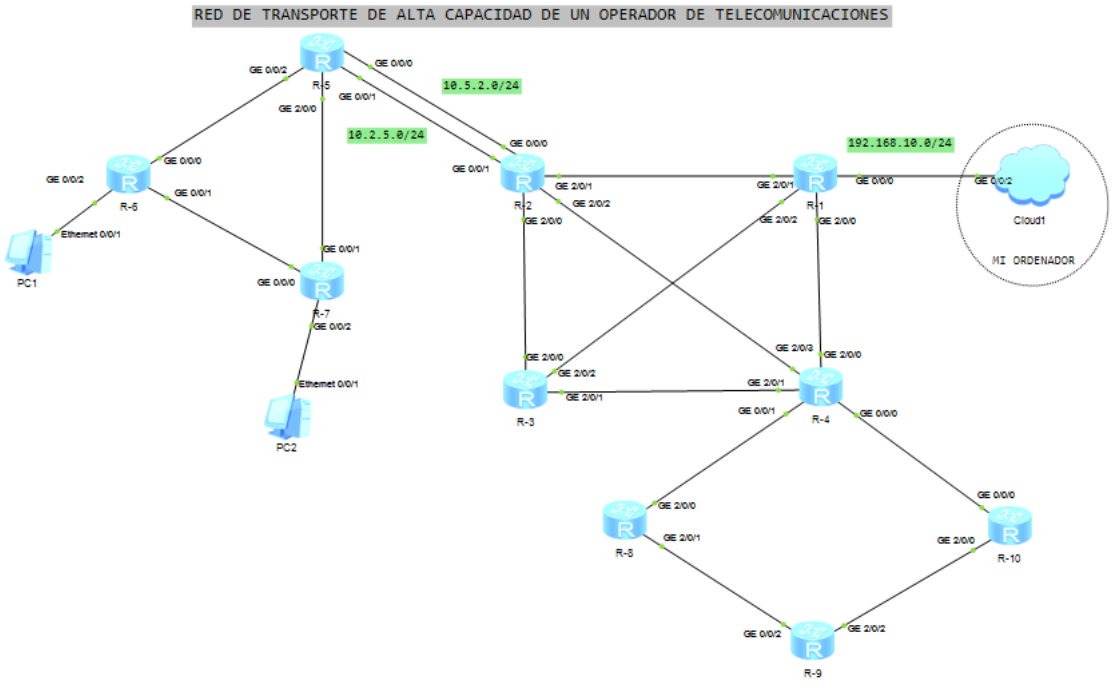


Figura 4.1: *Red desplegada en eNSP (2).*

Desde PC1 hacemos *ping* a R-10 y desde PC2 a R-1 para comprobar que existen comunicación entre los extremos. También hacemos *tracert* para conocer el camino que siguen los

paquetes. Hemos impuesto un costo a la interfaz GigabitEthernet0/0/0 en R-5 de 60 (y a la GigabitEthernet0/0/0 en R-2), para que al ser mayor que la de GigabitEthernet0/0/1 (valor 1 por defecto), escoja esta última como su ruta.

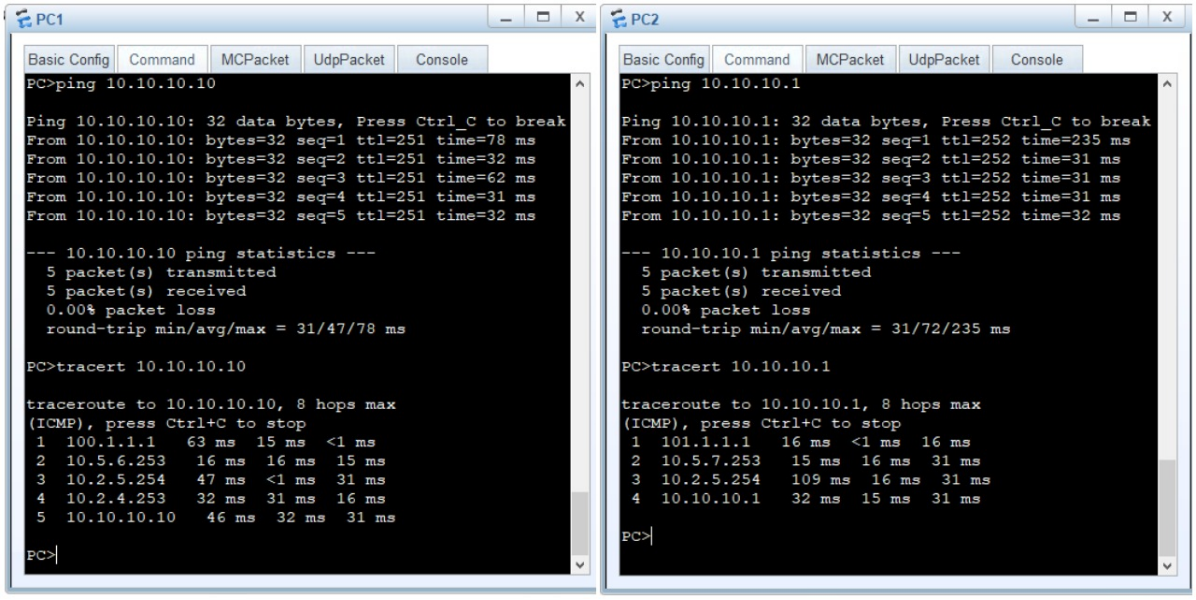


Figura 4.2: *Comandos de estado desde PC1 y PC2.*

Como se aprecia en la figura 4.2 ambos comandos de estado tienen resultados satisfactorios. De esta manera confirmamos que la red permite la comunicación extremo a extremo.

El dispositivo Cloud1 puede simular entre otras cosas un ordenador personal y de esta manera conectarlo con la topología en eNSP siendo un equipo más en la red. Esto se hace por medio de un adaptador virtual. En este caso el adaptador de bucle invertido de Windows. La configuración de Cloud1 la podemos ver en la figura 4.3.

En este punto tenemos una red simulada en eNSP en el mismo ordenador desde el cual nos conectaremos a ella. Ahora es necesario configurar una ruta estática en el ordenador, para alcanzar la red ya que no es accesible con la ruta de la puerta de enlace predeterminada. Esto lo hacemos desde el intérprete de comandos (cmd) en Windows.

En consola ejecutamos *route add [destino] mask [mascara_destino] [próximo_salto]*, para

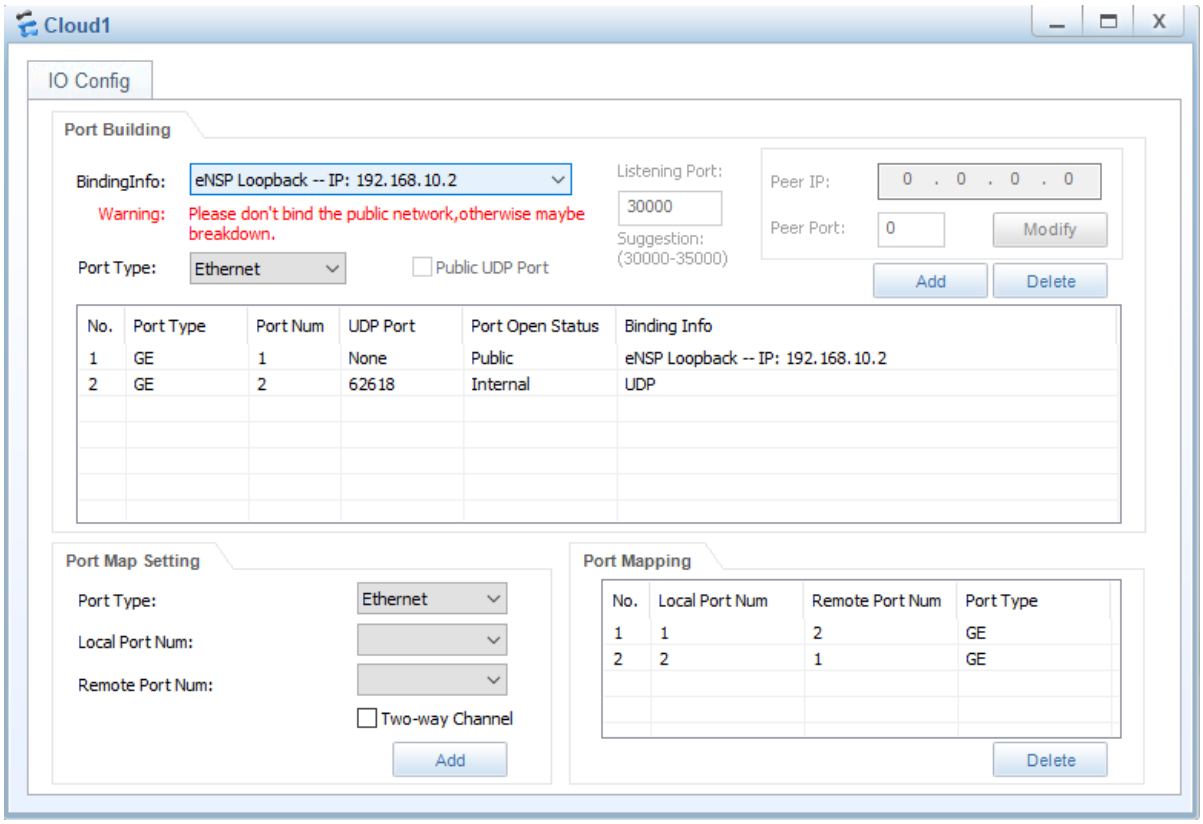


Figura 4.3: *Configuración para conectar el ordenador personal a eNSP.*

cada una de las interfaces de Loopback0 de la red, a fin de cuentas es esta por la cual haremos la conexión con SSH.

Comprobamos que desde el ordenador con comandos de estado podemos llegar a los *routers* de la red. Hacemos *ping* y *tracert* a R-7 y a R-8, y el resultado lo vemos en la figura 4.4.

Hasta aquí tenemos todas las condiciones creadas para hacer funcionar nuestra aplicación. El siguiente paso sería ejecutar los *scripts load_inventory.py* y *load_realtime.py* ya sea directamente o programarlo media tareas del sistema operativo. Para agilizar este proceso los ejecutaremos desde la consola de Ubuntu de Windows, para así también poder observar los mensajes de estado que incorporamos al código a medida que va avanzando.

La ejecución de *load_inventory.py* demora unos 45 segundos. Los mensajes de estado nos indican que tal y como lo programamos, se elimina primeramente la información de toda la

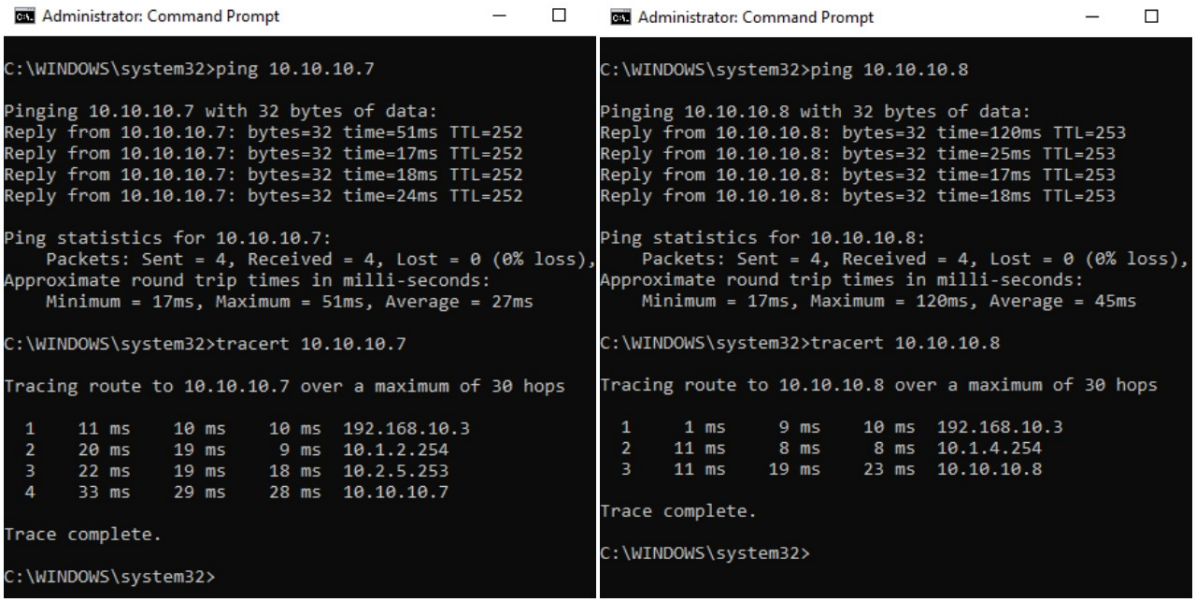


Figura 4.4: *Comandos de estado desde el ordenador a R-7 y R-8.*

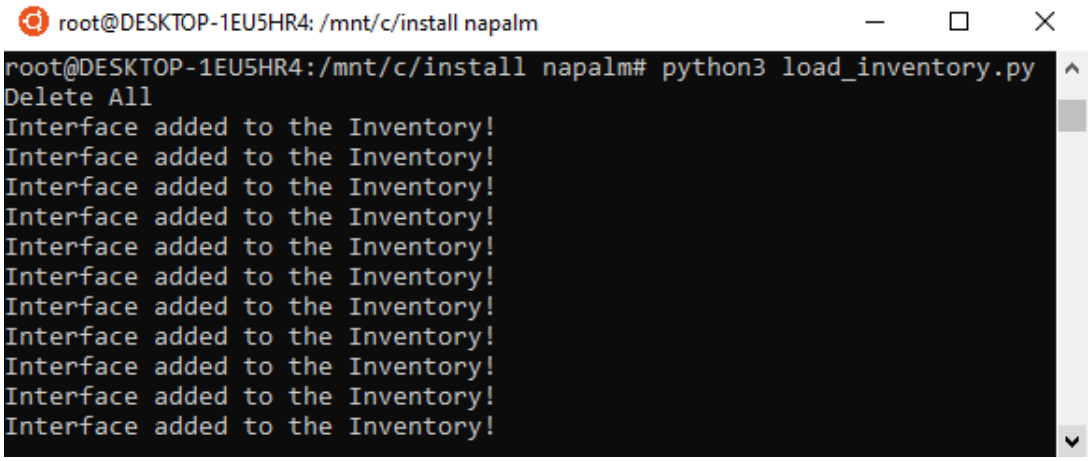


Figura 4.5: *Fragmentos de la ejecución de load_inventory desde la consola de Ubuntu.*

tabla en la base de datos y luego se añade cada fila del archivo Excel de inventario como una nueva entrada en *interfaces_inv*. Cuando en pantalla se imprime “FINISH” indica que el proceso ha terminado.

Tiempo entonces para el *load_realtime.py*. Este demora un poco más en completarse alrededor de unos 5 minutos. También tiene mensajes de estado según se ejecutan las principales acciones. Con un “FINISH” habrá terminado el proceso, y con él ambas tablas de la base de

datos tendrán la información actualizada. La aplicación web se encargará de utilizarla para mostrar ambas topologías.

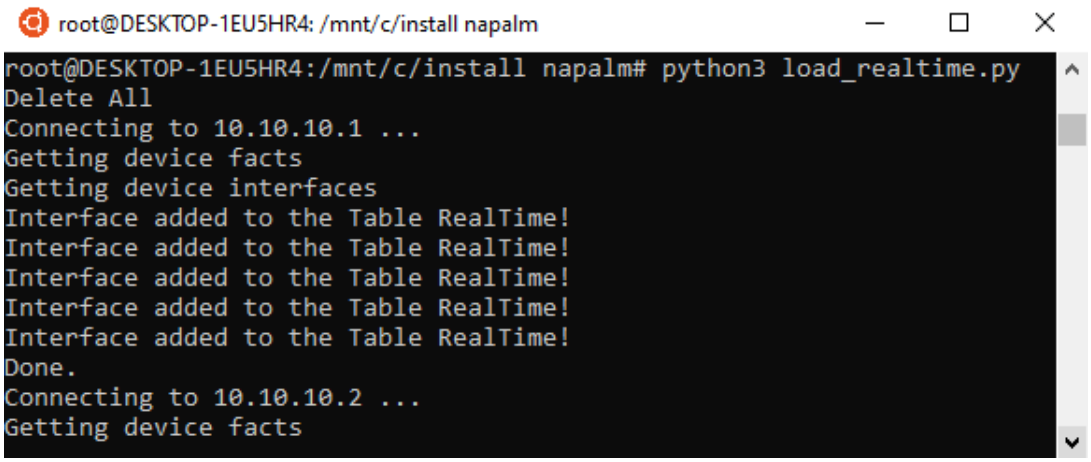


Figura 4.6: *Fragmentos de la ejecución de load_realtime desde la consola de Ubuntu.*

vendor	hostname	ubication	interface	ipv4	mask	is_up	is_enable	mac
Huawei	R-1	41.5193, 2.0776	LoopBack0	10.10.10.1	32	0	0	
Huawei	R-1		GigabitEthernet0/0/0	192.168.10.3	24	0	0	00:E0:FC:F0:49:13
Huawei	R-1		GigabitEthernet2/0/0	10.1.4.253	30	0	0	00:E0:FC:F3:49:13
Huawei	R-1		GigabitEthernet2/0/1	10.1.2.253	30	0	0	00:E0:FC:F3:49:14
Huawei	R-1		GigabitEthernet2/0/2	10.1.3.253	30	0	0	00:E0:FC:F3:49:15
Huawei	R-2	41.8024, -1.0318	LoopBack0	10.10.10.2	32	0	0	
Huawei	R-2		GigabitEthernet0/0/0	10.5.2.254	30	0	0	00:E0:FC:9A:4D:28

vendor	hostname	model	ubication	uptime	serial_num	interface	ipv4	mask	is_up	is_en	mac
Huawei	R-1	Unknown		-1	Unknown	GigabitEthernet0/0/0	192.168.10.3	24	1	1	00:E0:FC:F0...
Huawei	R-1	Unknown		-1	Unknown	GigabitEthernet2/0/0	10.1.4.253	30	1	1	00:E0:FC:F3...
Huawei	R-1	Unknown		-1	Unknown	GigabitEthernet2/0/1	10.1.2.253	30	1	1	00:E0:FC:F3...
Huawei	R-1	Unknown		-1	Unknown	GigabitEthernet2/0/2	10.1.3.253	30	1	1	00:E0:FC:F3...
Huawei	R-1	Unknown	41.5193, 2....	-1	Unknown	LoopBack0	10.10.10.1	32	1	1	
Huawei	R-2	Unknown		-1	Unknown	GigabitEthernet0/0/0	10.5.2.254	30	1	1	00:E0:FC:9...
Huawei	R-2	Unknown		-1	Unknown	GigabitEthernet0/0/1	10.2.5.254	30	1	1	00:E0:FC:9...
Huawei	R-2	Unknown		-1	Unknown	GigabitEthernet2/0/0	10.2.3.254	30	1	1	00:E0:FC:9...

Figura 4.7: *Estado de las tablas de la base de datos luego de ejecutar los scripts.*

En la figura 4.7 se puede ver un fragmento de como quedan rellenadas las tablas de la base de datos. En la parte superior la tabla *interfaces_inv* que guarda el inventario, debajo de ella la *interfaces_realtime* para la red en tiempo real. Para esta última, sucede que hay campos que tienen valores como es el caso de “Unknown” o “-1”, esto es así porque al no ser equipos reales, sino que están siendo simulados, algunos atributos toman valores desconocidos.

A continuación presentamos varias capturas de la aplicación web Napalm-App, que es la salida final y parte gráfica de nuestro proyecto.

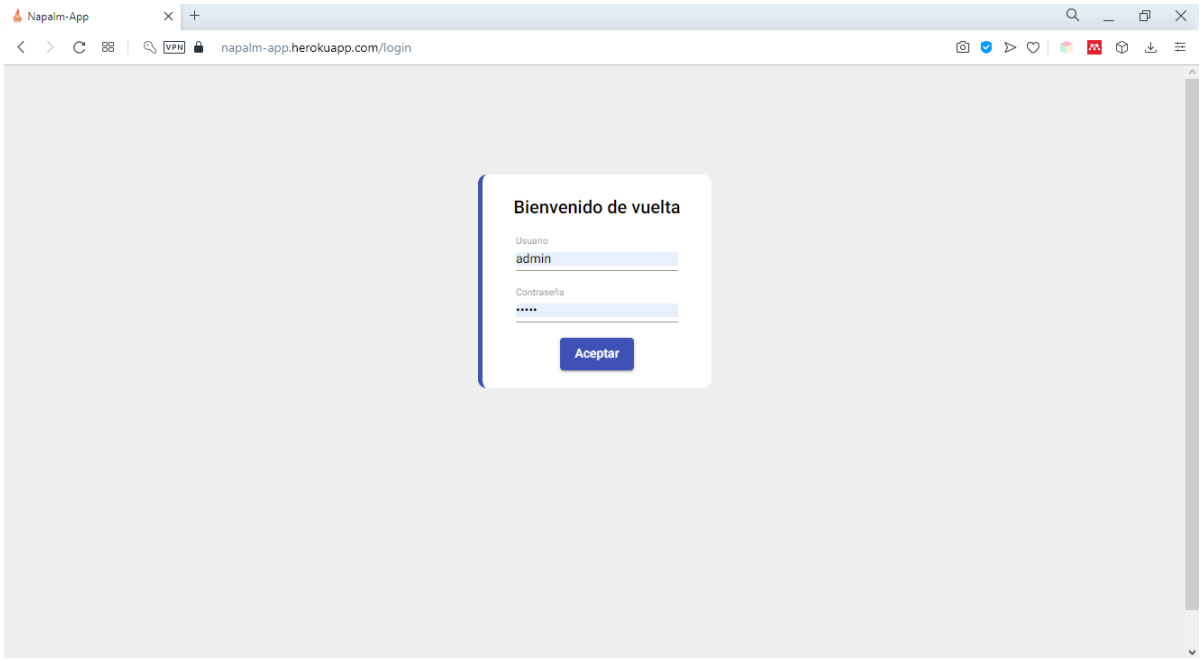


Figura 4.8: *Vista de login de Napalm-App.*

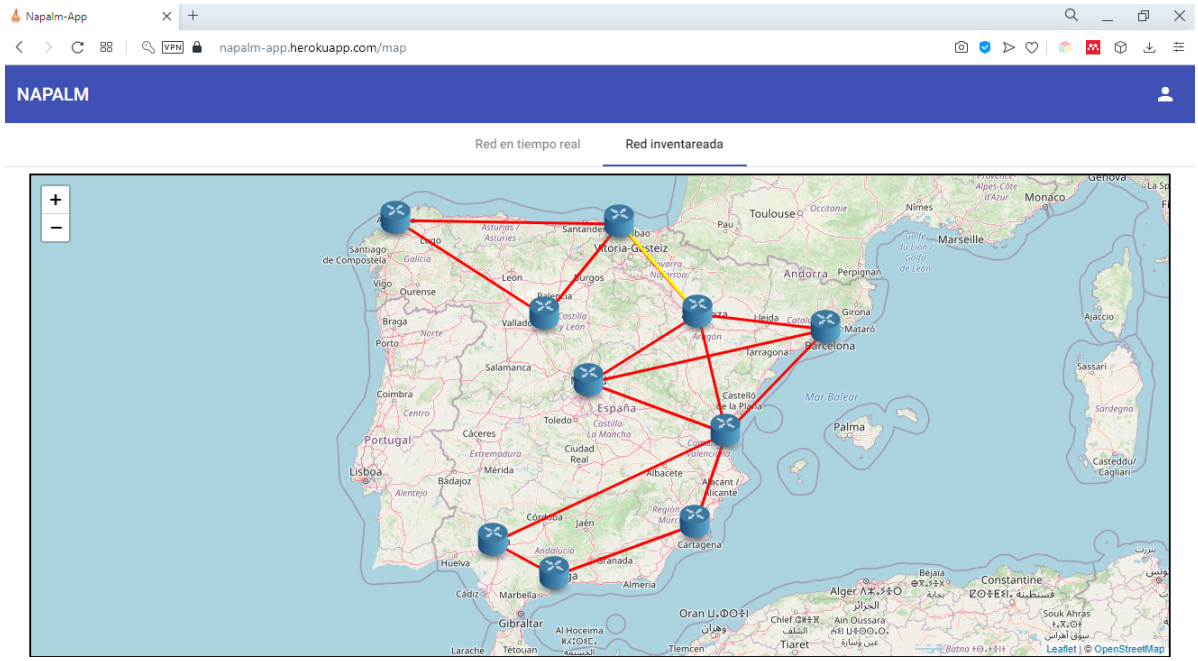


Figura 4.9: *Topología reconstruida para la red en inventario.*

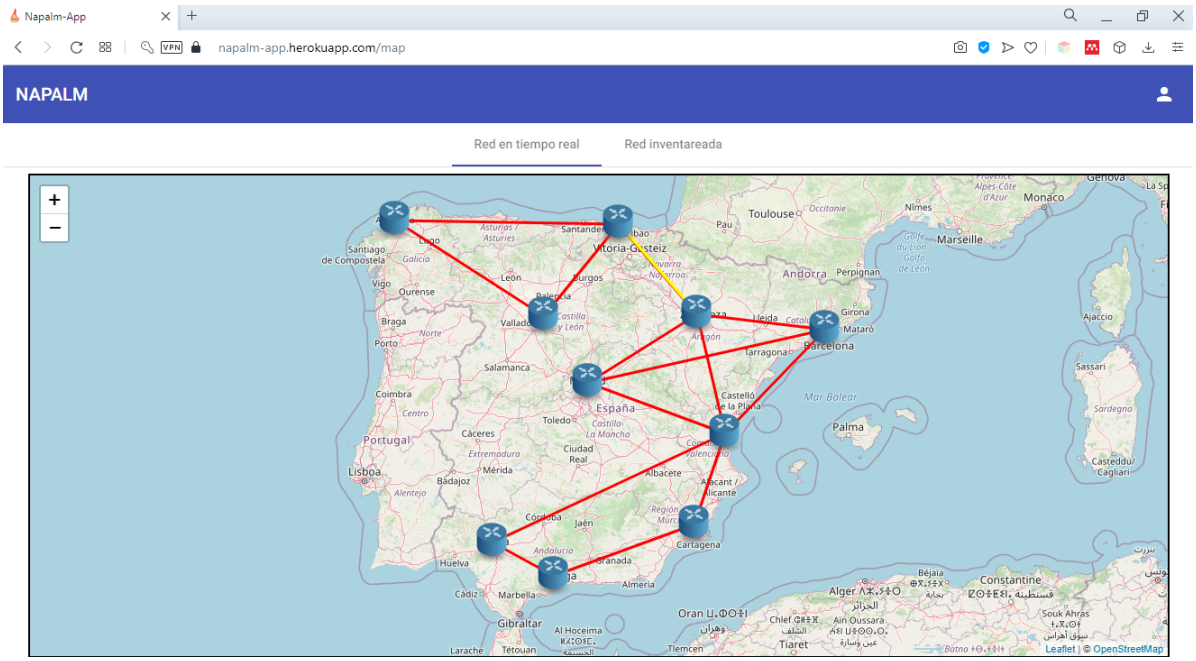


Figura 4.10: *Topología reconstruida para la red en tiempo real.*

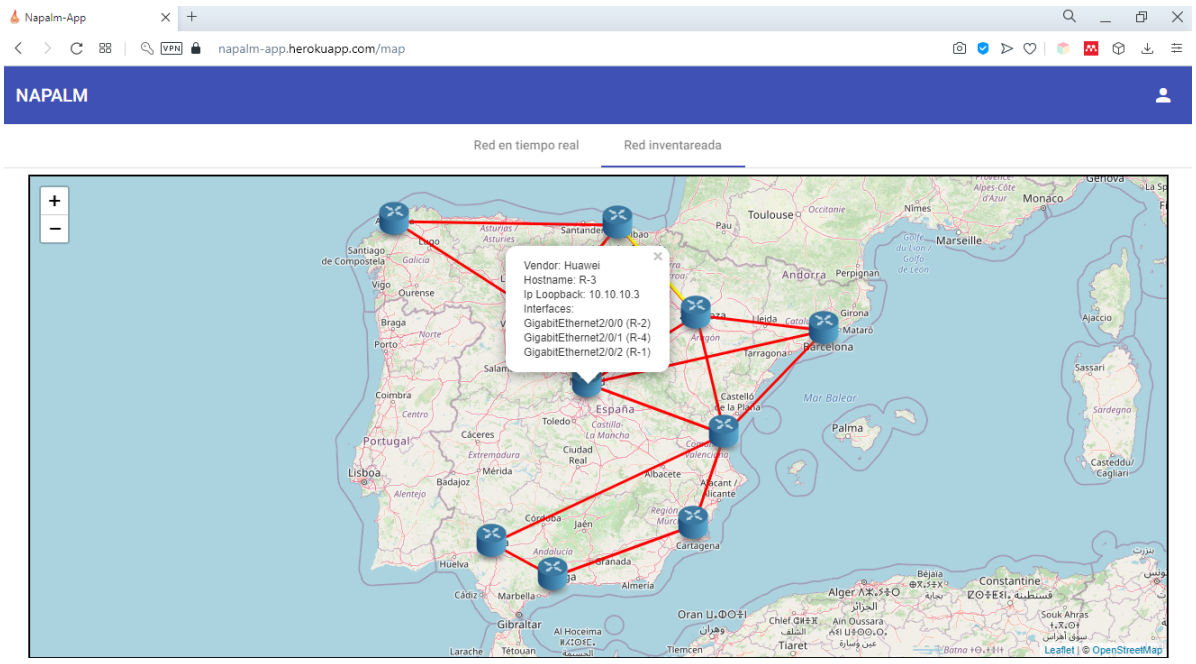


Figura 4.11: *Información adicional en los equipos de la red inventariada.*

En la topología reconstruida se aprecia que en uno de los enlaces se junta al color rojo de la línea, con otra línea amarilla.

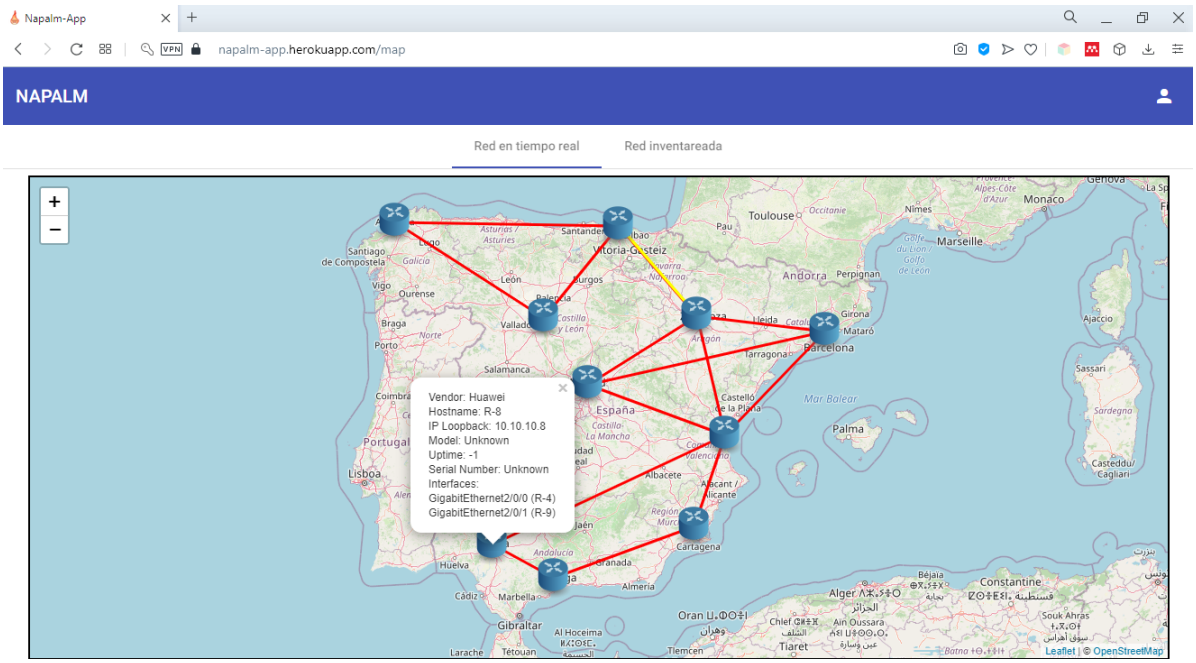


Figura 4.12: *Información adicional en los equipos de la red en tiempo real.*

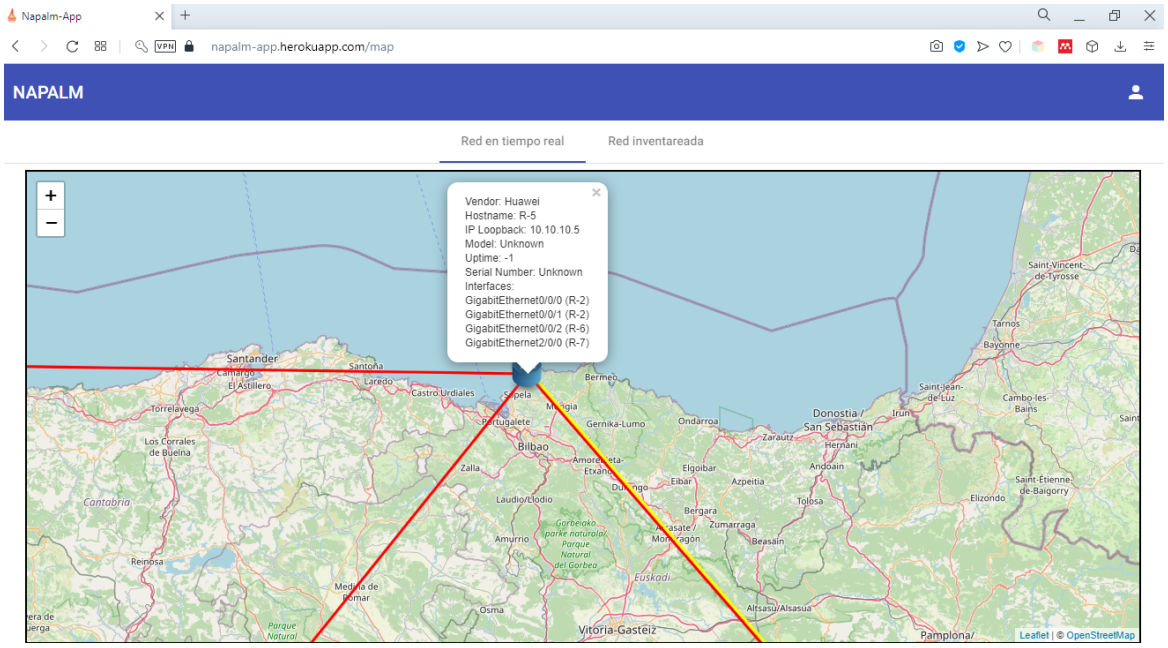


Figura 4.13: *Resultado de hacer zoom para ver el doble enlace.*

El trazo de color amarillo es para resaltar que existe un doble enlace entre los dos dispositivos ya que si ambos fueran rojos, desde una perspectiva sin hacer *zoom* sería imposible distinguirlos.

Así, le asignamos más colores en caso de más enlaces entre dos equipos (naranja, azul, violeta). La figura 4.13 nos muestra que es posible distinguirlos cuando hacemos *zoom* usando el color amarillo.

Si seleccionamos un equipo por medio de un *click*, se mostrará una pequeña ventana con información adicional. Dentro de esta, tenemos el nombre de cada interfaz y a su lado el equipo que conecta. La otra información disponible es de carácter general, pero sirven para demostrar todo lo que se puede explotar este tipo de aplicación.

Para comprobar el correcto funcionamiento, desde eNSP introducimos posibles fallas en la red en tiempo real. Apagamos interfaces y equipos para ver el cambio en la aplicación en la red en tiempo real y cómo en la inventariada sigue sin ninguna eventualidad. Para esto, luego de introducir el fallo debemos ejecutar *interfaces_inv* y *load_realtime.py*, aunque el primero puede llegar a ser innecesario ya que los cambios solo se registrarán con el segundo.

En las figuras 4.14 y 4.15 se aprecia el efecto de ejecutar los *scripts* luego de introducir los fallos.

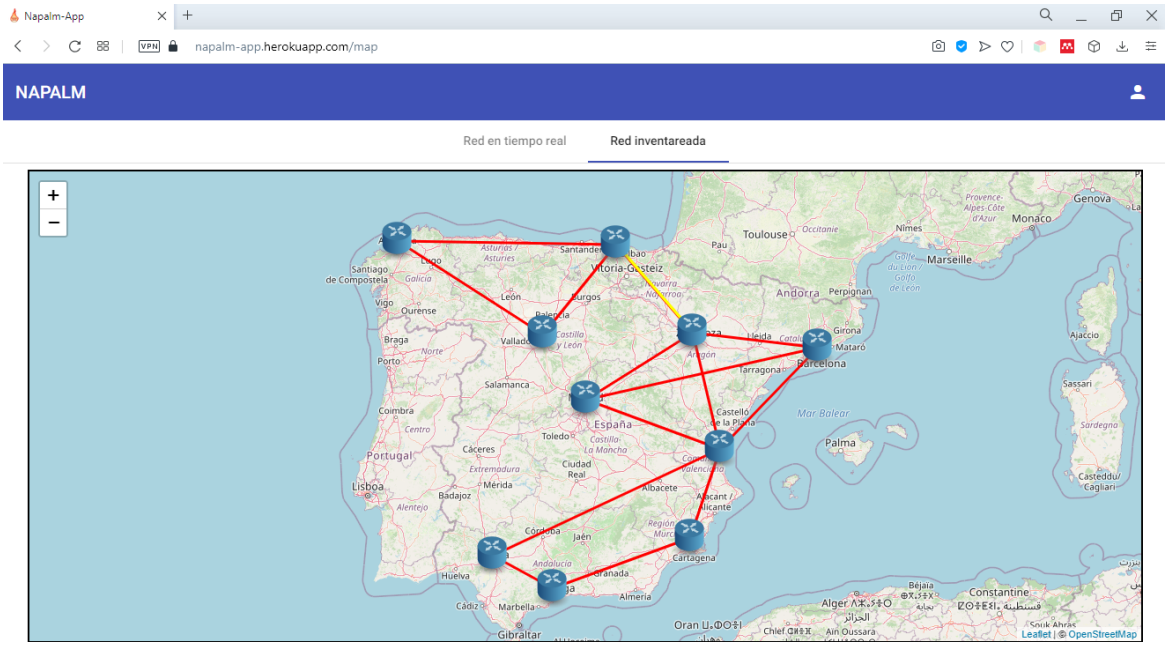


Figura 4.14: *Red en inventario luego de introducir los fallos.*

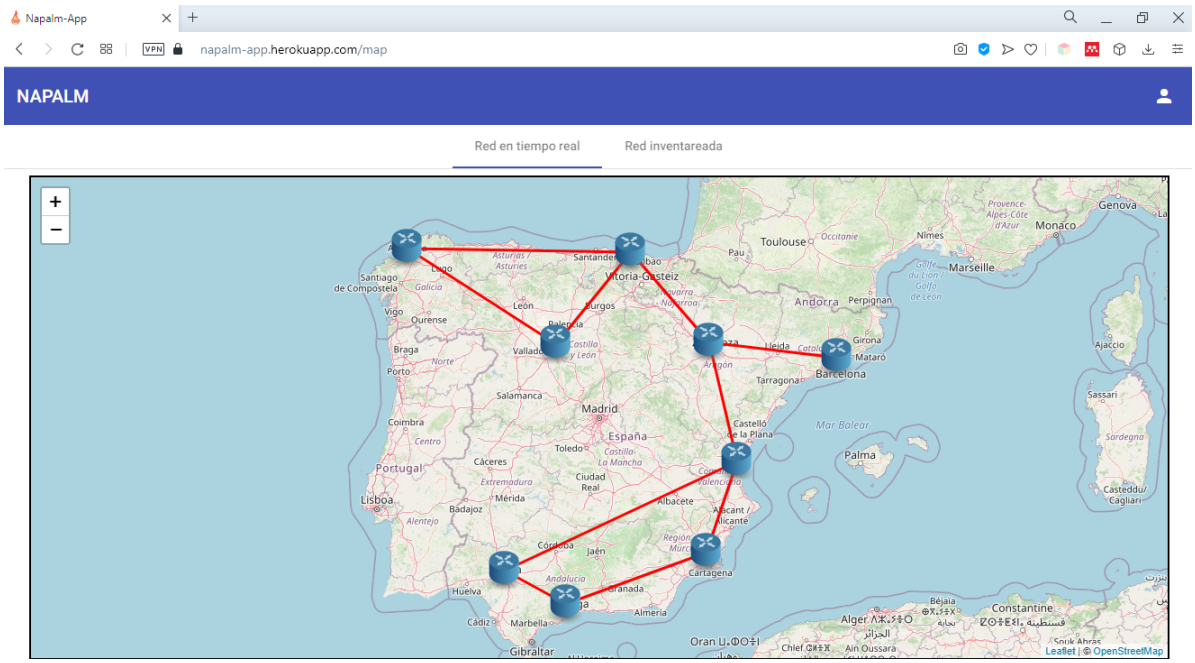


Figura 4.15: *Red en tiempo real luego de introducir los fallos.*

Como era de esperar la red en inventario no sufre ningún cambio. Sin embargo en la red en tiempo real se aprecia que falta un enlace entre Bilbao y Zaragoza (falta la línea amarilla), el *router* en Madrid lo hemos apagado y por tanto no sale en la topología, el enlace directo entre Barcelona y Valencia tampoco está. Los fallos en la red que se introducen desde eNSP se manifiestan en la herramienta web.

5 Capítulo 5: Conclusiones

En este trabajo se han dado elementos suficientes para otorgarle suma importancia a las herramientas de gestión y monitorización para un operador de redes de telecomunicaciones. Dentro de estos sistemas, los módulos del inventario cumplen un papel primordial ya que son el estado deseado de esta infraestructura. Por ejemplo, en el proceso de operación, surgen determinadas situaciones a modo de fallos que empujan a la red a eventos no planificados como sobrecargas de un enlace, congestión en el tráfico, mal funcionamiento de los servicios de la capa de aplicación, entre otros. Todos estos no hacen más que afectar al usuario final. Muchas de estas situaciones pueden resolverse si se cuenta con una herramienta visual que ayude a mostrar las conexiones en la red ya que la gran cantidad de los problemas son consecuencia de un cambio repentino en la topología física.

Es por ello que nos planteamos una serie de objetivos a cumplir en este trabajo para así conseguir una herramienta capaz de reconstruir la topología de red a partir de la configuración de los elementos. Para este caso, debía de ser una aplicación diferente a lo que se venía realizando en este campo, utilizamos una herramienta de código abierto que por demás nos brinda la posibilidad de mediante las mismas instrucciones, acceder a la configuración de cualquier fabricante de equipos. Sin lugar a dudas un enfoque novedoso en la materia.

Aseguramos que durante el desarrollo de este proyecto se han cumplido con éxito todos los objetivos planteados al inicio. Esto no obstante a que no fueron pocos los problemas que surgieron en el camino. No poder tener accesos a una red real, la imposibilidad de probar el código, al menos, para dos tecnologías a la vez y tener que ajustar el inventario de red a una red

desplegada en un simulador, no mermaron la consecución de las metas de nuestro trabajo. Poco a poco se fueron tomando decisiones de diseño tal cual sería el trabajo real de un ingeniero.

No se puede decir que por si solo una herramienta como esta resuelva todos los problemas y alteraciones que se producen en una red ya que la casuística puede ser muy variada. En ocasiones se hace necesario reconfigurar los equipos y los protocolos que en ellos están funcionando, o sea que para dar solución al problema se necesitarían aplicar criterios de ingeniería de tráfico, por citar un ejemplo.

Respecto a los programas en Python decir que se ha utilizado el lenguaje *de facto* en el desarrollo de tareas de automatización de red. Python y las bibliotecas que hacen uso de él, están ampliamente extendidas en las comunidad de desarrolladores y administradores de redes. Una de ellas, NAPALM, eje central de nuestro trabajo se va imponiendo debido a la capa de abstracción que brinda para el trabajo con múltiples tecnologías.

Por otra parte, el trabajo realizado guarda profunda relación con materias y conceptos estudiados a lo largo del máster en asignaturas tales como Redes de Nueva Generación, Diseño e Instalación de Redes de Telecomunicaciones y Arquitecturas Orientadas a Servicios para la Gestión de Contenidos. Es por ello que ha servido para tener una visión integral de un problema especifico en la vida laboral de un ingeniero en telecomunicaciones.

Para trabajos futuros se recomienda comprobar el funcionamiento con dos software de simulación de fabricantes diferentes, pero que estén conectados entre sí. Además sería interesante estudiar y extender el uso de NAPALM en el trabajo con otros protocolos en la medida de lo posible, como por ejemplo con MPLS (*Multiprotocol Label Switching*) en la reconstrucción de los LSP (*Label Switched Path*) o lo que es equivalente decir, en los caminos de intercambio de etiquetas.

Este trabajo, más que nunca, nos ha demostrado que con esfuerzo es posible encontrar una solución a los problemas que día a día nos surgen e intentan socavar el camino a seguir.

Bibliografía

[1] Rand Edwards. History and status of operations support systems. *Journal of Network and Systems Management*, 15(4):555–567, October 2007.

[2] Miyuki Sato. Creating next generation cloud computing based network services and the contributions of social cloud operation support system (OSS) to society. In *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. IEEE, June 2009.

[3] M.H. Sherif and S. Ho. Evolution of operation support systems in public data networks. In *Proceedings ISCC 2000. Fifth IEEE Symposium on Computers and Communications*. IEEE Comput. Soc.

[4] Guy Saadon, Yoram Haddad, and Noemie Simoni. A survey of application orchestration and OSS in next-generation network management. *Computer Standards & Interfaces*, 62:17–31, February 2019.

[5] Peter Busschbach, Steve Corum, Humberto La Roche, and Patrice Lamy. The role of management systems in optical/packet convergence. *Bell Labs Technical Journal*, 10(1):105–117, May 2005.

[6] SATEC. El OSS del futuro, 2021. `https://www.satec.es/blog/2021/01/13/el-oss-del-futuro`, accedido 2021-02-15.

[7] Tecsidel. Gestión de red (NMS) y soporte a la operación (OSS), 2014. https://tecsidel.com/press/gestion-de-red-nms-y-soporte-a-la-operacion-oss/, accedido 2021-02-15.

[8] Frederic Raspall. Building nemo, a system to monitor IP routing and traffic paths in real time. *Computer Networks*, 97:1–30, March 2016.

[9] Konstantinos Kotsopoulos. *Proceedings of the first ITU-T Kaleidoscope Academic Conference : innovations in NGN, Geneva, 12-13 May 2008*. International Telecommunication Union, Geneva, 2008.

[10] Charles Gillan, Peter Kilpatrick, Ivor T. A. Spence, T. John Brown, Rabih Bashroush, and Rachel Gawley. Challenges in the application of feature modelling in fixed line telecommunications. In Klaus Pohl, Patrick Heymans, Kyo Chul Kang, and Andreas Metzger, editors, *First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007, Limerick, Ireland, January 16-18, 2007. Proceedings*, volume 2007-01 of *Lero Technical Report*, pages 141–148, 2007.

[11] International Telecommunications Union. ITU-T M.3010 (02/2000) Principles for a telecommunications management network, 2000. http://handle.itu.int/11.1002/1000/4869, accedido 2021-02-21.

[12] Doris Wong Hooi Ten, Selvakumar Manickam, Sureswaran Ramadass, and Hussein A. Al Bazar. Study on advanced visualization tools in network monitoring platform. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*. IEEE, 2009.

[13] CH Lin, HC; Wang. *GLOBECOM 1999 : seamless interconnection for universal services : conference record, Global Telecommunications Conference, 5-9 December 1999, Rio de Janeiro, Brazil, Intercontinental and Sheraton Hotels*. IEEE Operations Center, Piscataway, NJ, 1999.

[14] FL Jiang, XP; Peng. *The First International Symposium on Computer Network and*

Multimedia Technology : CNMT 2009, Wuhan, China, 18-20 December 2009. IEEE, Piscataway, N.J, 2009.

[15] Bruce B. Lowekamp. Combining active and passive network measurements to build scalable monitoring systems on the grid. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):19–26, March 2003.

[16] Leon;Kong Yang; Brad Hale Adato. Network Monitoring For Dummies, 2016. http://images.solpub.com/Resources/Network_Monitoring_For_Dummies_SolarWinds_Special_Edition.pdf, accedido 2021-03-20.

[17] Hao Wang. Improvement and implementation of wireless network topology system based on SNMP protocol for router equipment. *Computer Communications*, 151:10–18, February 2020.

[18] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Madrid: Addison-Wesley, Reading, Mass, 2006.

[19] Carl Moberg. Understanding NETCONF and YANG, 2019. https://www.networkworld.com/article/2173842/understanding-netconf-and-yang.html, accedido 2021-02-05.

[20] RFC 8345 IETF. The Syslog Protocol, 2018. https://tools.ietf.org/html/rfc8345, accedido 2021-01-08.

[21] Cisco Networks. What Is Network Management?, 2019. https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-management.html#~q-a, accedido 2021-02-20.

[22] Wenwei Li, Dafang Zhang, Gaogang Xie, and Jinmin Yang. TCP and ICMP in network measurement: An experimental evaluation. In *Parallel and Distributed Processing and Applications*, pages 870–881. Springer Berlin Heidelberg, 2005.

[23] RFC 5424 IETF. The Syslog Protocol, 2018. https://datatracker.ietf.org/doc/rfc5424/, accedido 2021-02-05.

[24] Paessler. ¿Qué es Syslog?, 2019. https://www.es.paessler.com/it-explained/syslog, accedido 2020-02-05.

[25] Holman Rivera Linares, Fredy Silva Cubillos, Jairo Hernández Gutierrez, and Darin Mosquera Palacios. Gestión gráfica de dispositivos activos de red multivendedor. *Ingeniería Solidaria*, 14(24):1–11, January 2018.

[26] ijcrt.org. Network Monitoring Tools and Technologies, 2018. https://ijcrt.org/papers/IJCRT1812394.pdf, accedido 2021-01-08.

[27] Atul Kumar, Ayushman Antal, Gautam Kumar, and Abhineet Anand. Network monitoring and analytics management using web platform. 2017.

[28] IBM. Overview of IBM Tivoli Monitoring, 2021. https://www.ibm.com/docs/es/itcam-app-mgr/7.2.1?topic=i-overview-tivoli-monitoring-1, accedido 2021-01-08.

[29] Grupo Arión. Overview of IBM Tivoli Monitoring, 2020. https://www.grupoarion.com.mx/soluciones-bmc/truesight-monitoreo/, accedido 2021-01-03.

[30] Broadcom. Unified Infrastructure Management, 2021. https://www.broadcom.com/info/aiops/unified-infrastructure-management, accedido 2021-01-04.

[31] ManageEngine. ManageEngine OpManager, el software de monitoreo de red confiable, 2021. https://www.manageengine.com/es/network-monitoring/, accedido 2021-01-08.

[32] Imagine Comunications. Magellan Network Management System, 2021. https://imaginecommunications.com/product/

magellan-network-management-system/, accedido 2021-01-08.

[33] Forum Huawei. Comandos TL1 consola U2000/eSight, 2021. https://forum.huawei.com/enterprise/es/comandos-tl1-consola-u2000-esight-nbi/thread/693083-100241, accedido 2021-01-07.

[34] Mi-Jung Choi, Hong-Taek Ju, James W. Hong, and Dong-Sik Yun. Design and implementation of web services-based NGOSS technology-specific architecture. *annals of telecommunications - annales des télécommunications*, 63(3-4):195–206, February 2008.

[35] DNSStuff. What Is Network Topology? Best Guide to Types and Diagrams, 2019. https://www.dnsstuff.com/what-is-network-topology, accedido 2021-01-07.

[36] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous IP networks: The NetInventory system. *IEEE/ACM Transactions on Networking*, 12(3):401–414, June 2004.

[37] Andrew Tanenbaum. *Redes de computadoras*. Pearson Educación, México, 2012.

[38] CCNA Desde Cero. Topologías de Red. https://ccnadesdecero.com/blog/topologias, accedido 2021-06-15.

[39] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Deployment of an algorithm for large-scale topology discovery. *IEEE Journal on Selected Areas in Communications*, 24(12):2210–2220, December 2006.

[40] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery, 2000.

[41] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. In *In Proc. ACM SIGCOMM*, pages 133–145, 2002.

[42] Jangwon Lee and Gustavo De Veciana. Resource and topology discovery for ip multicast

using a fan-out decrement mechanism. In *in Proc. IEEE INFOCOM*, pages 1627–1635, 2001.

[43] Kurt Tutschku and Herbert Baier. Characterizing network performance for enterprise networks, 2001.

[44] Paul Mihăilă, Titus Bălan, Radu Curpen, and Florin Sandu. Network automation and abstraction using python programming methods. *MACRo 2015*, 2(1):95–103, October 2017.

[45] Jason Edelman. *Network programmability and automation : skills for the next-generation network engineer*. O'Reilly Media, Sebastopol, CA, 2018.

[46] Mircea Ulinic. Network Automation with Salt and NAPALM. `https://speakerdeck.com/mirceaulinic/network-automation-with-salt-and-napalm`, accedido 2021-06-15.

[47] Adian Fatchur Rochim, Abda Rafi, Adnan Fauzi, and Kurniawan Teguh Martono. AsRaD system as a design model of the network automation configuration system based on the REST-API and django framework. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, pages 291–298, November 2020.

[48] Adam Pavlidis, Marinos Dimolianis, Dimitris Kalogeras, and Vasilis Maglaris. Automated distribution of access control rules in defense layers of an enterprise network. In Joe Betser, Carol J. Fung, Alex Clemm, Jérôme François, and Shingo Ata, editors, *IFIP/IEEE International Symposium on Integrated Network Management, IM 2019, Washington, DC, USA, April 09-11, 2019*, pages 735–736. IFIP, 2019.

[49] NAPALM. NAPALM Network Automation for the people, by the people, 2021. `https://napalm-automation.net`, accedido 2021-01-07.

[50] Adam Pavlidis. Network Automation and Orchestration with Saltstack. https://slideplayer.com/slide/17091120, accedido 2021-06-15.

[51] Kirk Byers David Barroso, Mircea Ulinic. NAPALM's documentation. https://napalm.readthedocs.io/en/latest/base.html, accedido 2021-01-07.

[52] redeszone. Lista de simuladores de redes para virtualizar nuestra propia red, 2014. https://www.redeszone.net/2014/03/20/lista-de-simuladores-de-redes-para-virtualizar-nuestra-propia-red/, accedido 2021-01-07.

[53] Mikrotik. Manual Winbox, 2020. https://wiki.mikrotik.com/wiki/Manual:Winbox, accedido 2021-01-07.

[54] Huawei. Introducción al simulador de red eNSP, 2019. https://forum.huawei.com/enterprise/es/introducción-al-simulador-de-red-de-huawei-ensp, accedido 2021-01-07.

[55] Huawei. What Is OSPF and How Is It Configured? https://support.huawei.com/enterprise/en/doc/EDOC1100082074, accedido 2021-03-07.

[56] BBVA ApiMarket. API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos. https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/, accedido 2021-03-07.

[57] Heroku. Web Oficial. https://www.heroku.com, accedido 2021-06-08.

[58] Google Angular. Web Oficial. https://angular.io, accedido 2021-06-12.

[59] Leaflet. Web Oficial. https://leafletjs.com, accedido 2021-06-12.